Proof Nets as Processes

Dimitris Mostrous

DI–FCUL–TR–2012–07

DOI:10455/6890

(http://hdl.handle.net/10455/6890)

October 2012

Proof Nets as Processes

Dimitris Mostrous

Department of Informatics, University of Lisbon, Portugal
dimitris@di.fc.ul.pt

Abstract. We present delta-calculus, a novel interpretation of Linear Logic, in the form of a typed process algebra that enjoys a Curry-Howard correspondence with Proof Nets. Reduction inherits the qualities of the logical objects: termination, deadlock-freedom, determinism, and very importantly, a high degree of parallelism. We obtain the necessary soundness results and provide a propositions-as-types theorem. The basic system is extended in two directions. First, we adapt it to interpret Affine Logic. Second, we propose extensions for general recursion, and introduce a novel form of recursive linear types. As an application we show a highly parallel type-preserving translation from a linear System F and extend it to the recursive variation. Our interpretation can be seen as a more canonical proof-theoretic alternative to several recent works on pi-calculus interpretations of linear sequent proofs (propositions-as-sessions) which exhibit reduced parallelism.

1 Introduction

Linear Logic has had a profound impact in both programming and proof theory, and has been the subject of many computational interpretations starting from Abramsky's seminal work [2]. Proof Nets [21,23], the canonical proof objects of Linear Logic, are an abstract, geometrical representation of sequent proofs that is inherently parallel due to locality, or context-independence, of many normalising steps. They enjoy several important properties, in particular determinism (semantic confluence), termination (strong normalisation), and consequently deadlock-freedom. Such characteristics are highly desirable, and can inform the compilation of functional programs for multicore execution, without running the risk of returning the wrong result, of deadlock, or of unintended divergence. In many domains, such as code mobility, operating systems, and critical software, these facets are of paramount importance.

In a recent development, Caires and Pfenning [11] presented an interpretation of sequent proofs of (dual) intuitionistic linear logic as typed terms of the synchronous π-calculus, and proved termination [37]. Their system was subsequently adapted to a classical linear logic setting by Wadler [41]. However, π-calculus imposes sequentiality that impedes parallelism, so is not the best choice as a language for linear proofs. For example, in $x(y).z(k).P \mid \overline{x}\langle a\rangle \mid \overline{z}\langle b\rangle$, z may be safe to reduce, but is blocked under $x(y)$. Even if $x(y).z(k).P \equiv z(k).x(y).P$ was allowed, x would be guarded, so the two redices cannot be performed in parallel.

We developed an interpretation of proofs of the (classical) linear sequent calculus into a new process language, called δ-calculus, that corresponds to proof nets, and is typed with linear formulas. The multiplicative ($\otimes/\parr$) fragment, derived *directly* from a process interpretation of proof nets, can be seen as a linear variation of Solos [31] with a twist: axioms are interpreted as *explicit substitutions*. Specifically, an axiom ab behaves like an explicit *fusion* [20] $a{=}b$, *e.g.*, $(\boldsymbol{\nu} x)\,(ax \mid xb) \rightsquigarrow^2 ab$. Such a term cannot reduce in Solos (see [31]).

δ-calculus can be utilised to study proof theory in a process setting, for parallel compilation of functions and (functional) objects, and for programming in the style of sessions (as in [11], but with more asynchrony). We interpret full Second-order Linear Logic (with Mix), omitting the uninteresting additive units which are, computationally, of limited use (Section 2). Two useful extensions are detailed, that go beyond the results of the aforementioned systems. First, we present an adaptation that interprets Affine (Linear) Logic (Section 2), and second, we extend proof net exponentials (i.e., the terms than can be reused) to facilitate general recursion, adding an original form of *linear recursive types* (Section 3). Affine terms, used *at most once*, allow more liberal implementations of typed interactions where protocols can be safely *cancelled* at any point. Recursion allows to encode more realistic programming idioms at the expense of logical consistency. Nevertheless, even the strongly normalising fragment supports primitive recursion and the definition of the usual data types [21].

We state the basic soundness properties, followed by a Curry-Howard correspondence and a progress theorem for recursive nets (Section 4). To illustrate the practical aspects of our formulation, we provide a remarkably parallel, type-preserving translation from a *Linear* System F into δ-calculus nets, allowing programs (or proofs) to be written in a more traditional syntax (Section 5). With recursion, we can also encode fixpoints, leading to more powerful programming facilities (also in Section 5). We conclude by discussing related work (Section 6).

2 Proof Nets as Processes

A proof net consists of a collection of *links*, connectives which have premises and conclusions. The links are *wired* together, for example a conclusion of one can be a premise of another, and the whole composition forms a graph called a *proof structure*. When a proof structure corresponds to a sequent derivation, it is called a *proof net*, and the link conclusions which are not wired as premises to other links provide the conclusions of the sequent derivation. The *correctness criterion*, a purely geometrical method by which this correspondence is checked, was originally defined by Girard [21] and simplified later by Danos and Regnier [14] (for a fragment of the logic). In this work we check correctness by typing, but note that the complexity of the criterion and that of type-checking may be related. Proof net normalisation [21,22] amounts to the (deterministic) re-linking of the uniquely (but implicitly) identified parts of a net: once we assign *names* to logical connectives, we obtain a *process algebra* which we call δ-calculus.

Types and Duality We begin by presenting the syntax of linear logic types:

$$A, B, C \ ::= \ A \otimes B \mid A \parr B \mid \mathbf{1} \mid \bot \mid A \,\&\, B \mid A \oplus B \mid !A \mid ?A \mid \forall \mathsf{X}.A \mid \exists \mathsf{X}.A \mid \mathsf{X} \mid \mathsf{X}^{\bot}$$

for tensor, par, tensor and par units, with, plus, of course, why not, for all, exists, type variables. Duality (negation) is defined as follows, playing a role familiar from session types: it allows complementary communications to take place.

$$\mathbf{1}^{\bot} \stackrel{\mathsf{def}}{=} \bot \qquad \bot^{\bot} \stackrel{\mathsf{def}}{=} \mathbf{1} \qquad (!A)^{\bot} \stackrel{\mathsf{def}}{=} ?A^{\bot} \qquad (?A)^{\bot} \stackrel{\mathsf{def}}{=} !A^{\bot} \qquad (A \otimes B)^{\bot} \stackrel{\mathsf{def}}{=} A^{\bot} \parr B^{\bot}$$
$$(A \parr B)^{\bot} \stackrel{\mathsf{def}}{=} A^{\bot} \otimes B^{\bot} \qquad (A \,\&\, B)^{\bot} \stackrel{\mathsf{def}}{=} A^{\bot} \oplus B^{\bot} \qquad (A \oplus B)^{\bot} \stackrel{\mathsf{def}}{=} A^{\bot} \,\&\, B^{\bot}$$
$$(\forall \mathsf{X}.A)^{\bot} \stackrel{\mathsf{def}}{=} \exists \mathsf{X}.A^{\bot} \qquad (\exists \mathsf{X}.A)^{\bot} \stackrel{\mathsf{def}}{=} \forall \mathsf{X}.A^{\bot} \qquad (\mathsf{X}^{\bot})^{\bot} \stackrel{\mathsf{def}}{=} \mathsf{X}$$

The Multiplicative Fragment We start by presenting the fragment corresponding to Multiplicative Linear Logic (MLL), to elucidate the main concepts upon which the remaining extensions towards (and beyond) full Linear Logic can be understood easily. The language is explicitly typed, but for brevity we omit types in the reduction semantics. The syntax of terms π, $\rho \ldots$ is shown below:

$\pi, \rho ::=$	(net)	$\mathsf{L}(a) ::=$	(link)	
$\mathsf{L}(a)$	(link)	ab	(axiom)	
$(\boldsymbol{\nu} a\colon A)\,\pi$	(scope)	$\overline{a}(b,c)$	(tensor $\star$)	
$\sim a$	(clean)	$a(b,c)$	(par)	
$\pi \mid \rho$	(parallel)	$\overline{a}$	(one $\star$)	($\star$) *marks a link as protected,*
$\boldsymbol{\epsilon}$	(nil)	a	(bottom $\star$)	*defining the set of $\star$-links.*

We assume a countable set of names a, b, x, y etc., the usual notions of free and bound names, and α-conversion. $\mathsf{L}(a)$ stands for a link (i.e., a logical connective) with conclusion a. We mark some links with ($\star$): these are called *protected*, for reasons explained later. All names appearing in links are free, and are only made bound with scope restriction or one of the other forms which we subsequently explain. Parallel composition is n-ary commutative with unit $\boldsymbol{\epsilon}$ (to avoid confusion with the additive unit $\mathbf{0}$ of Linear Logic). Association priority is given by $\cdot$, $\boldsymbol{\nu}$, $\mid$, $+$, noting that boxing ($\mathbf{.}$) and summation ($+$) appear later. The unitary links $\overline{a}$ and a have one conclusion and no premise, the axiom ab has two distinct conclusions and no premise, and the tensor and par links, $\overline{a}(b,c)$ and $a(b,c)$, have one conclusion (a) and two premises (b and c). The choice of writing $\overline{a}$ in $\otimes$-links and a in $\parr$-links, and similarly throughout, is a mnemonic for *polarity* [5].

The reduction relation $\pi \rightsquigarrow \rho$ corresponds to the axiom and symmetric normalisation steps of proof nets [21], and uses evaluation contexts [18]. Although there are *no labels*, we make use of the convenient notation: $\pi \stackrel{a}{\rightsquigarrow} \rho \stackrel{\mathsf{def}}{=} \pi \rightsquigarrow \sim a \mid \rho$.

$$(\boldsymbol{\nu} a)\,(\sim a \mid \pi) \rightsquigarrow \pi \qquad \mathsf{L}(a) \mid ab \stackrel{a}{\rightsquigarrow} \mathsf{L}(b) \qquad (\sim\boldsymbol{\nu},\ \mathsf{Link}-\mathsf{Ax})$$
$$a \mid \overline{a} \stackrel{a}{\rightsquigarrow} \boldsymbol{\epsilon} \qquad a(b,c) \mid \overline{a}(d,e) \stackrel{a}{\rightsquigarrow} bd \mid ce \qquad (\bot-\mathbf{1},\ \parr-\otimes)$$

$\mathsf{G}[\,\cdot\,] ::= (\boldsymbol{\nu} a)\,(\,\cdot\,)$ (extended later)

$\mathsf{C}[\,\cdot\,] ::= \ \cdot \ \mid \ (\mathsf{C}[\,\cdot\,] \mid \pi) \ \mid \ \mathsf{G}[\,\mathsf{C}[\,\cdot\,]\,]$

$$\frac{\pi \rightsquigarrow \rho}{\mathsf{C}[\,\pi\,] \rightsquigarrow \mathsf{C}[\,\rho\,]} \qquad \frac{\pi \equiv \pi' \rightsquigarrow \rho' \equiv \rho}{\pi \rightsquigarrow \rho} \quad (\mathsf{CR},\ \mathsf{CC})$$

A normalisation step takes place when two links react, generating new links so as to preserve the correctness of the structure. There is no cut link as in proof

nets, but following Abramsky's π-calculus translation (see Bellin and Scott [9]), a cut is implicitly defined between actions that share a name as their conclusion. Thus, in a well-typed proof structure, each name appears *once* (as conclusion) or *twice* (when two conclusions cut, or a conclusion is wired to a premise).

An axiom can be seen as an *explicit fusion* [36], or a symmetric adaptation of a *linear forwarder* [19], and contracts with any link with which it shares a conclusion, using $(\mathsf{Link}-\mathsf{Ax})$ which defines a family of rules for all $\mathsf{L}(a)$. For example, using (CR), $(\mathsf{Link}-\mathsf{Ax})$, and $(\sim\boldsymbol{\nu})$ we obtain the reduction $(\boldsymbol{\nu}x)\,(ax \mid xc) \rightsquigarrow (\boldsymbol{\nu}x)\,(\sim x \mid ac) \rightsquigarrow ac$. Notice how this has the effect of a local substitution, since there is no irrelevant context, which immediately increases the *parallelism* that can be attained. Recall also that this term cannot reduce in Solos (see the introduction to [31]) since monadic communication is not symmetric (i.e., $\overline{a}b \mid ax \rightarrow$ but $xa \mid \overline{a}b \not\rightarrow$). The residual $\sim x$ is released so as to interact (and this may be seen as another kind of cut) with the enclosing scope of $(\boldsymbol{\nu}x)$, using $(\sim\boldsymbol{\nu})$, in order to destroy it, since x is never reused in a linear setting. This can be understood as an asynchronous deallocation, decoupling reductions from their enclosing context. The symmetric contraction between a tensor and par link, with rule $(\parr-\otimes)$, produces two new axioms that turn the (passive) premises to (active) conclusions, which may now react. This reduction is completely local, and therefore can be performed in parallel to any other redex that may be enabled. We now define a useful abbreviation, used throughout the work:

$$\pi\{{}^{a}\!/\!_{x}\} \stackrel{\mathsf{def}}{=} (\boldsymbol{\nu}x)\,(ax \mid \pi) \qquad \textit{for example:} \quad xc\{{}^{a}\!/\!_{x}\} \stackrel{\mathsf{def}}{=} (\boldsymbol{\nu}x)\,(ax \mid xc) \rightsquigarrow^{2} ac$$

As a logical system, proof nets enjoy cut-elimination (strong normalisation), which is achieved with commutative contractions [21], subsumed by $\equiv$ (cf. (CC)).

$$\frac{\pi \equiv \rho}{\mathsf{C}[\,\pi\,] \equiv \mathsf{C}[\,\rho\,]} \qquad ab \equiv ba \qquad \pi \mid \rho \equiv \rho \mid \pi \qquad \pi \equiv \pi \mid \boldsymbol{\epsilon} \qquad (\boldsymbol{\nu}a)(\boldsymbol{\nu}b)\,\pi \equiv (\boldsymbol{\nu}b)(\boldsymbol{\nu}a)\,\pi \qquad (\mathsf{Standard}, \equiv \boldsymbol{\nu}\boldsymbol{\nu})$$

$$\mathsf{G}[\,(\boldsymbol{\nu}a)\,(\pi \mid \rho)\,] \equiv \mathsf{G}[\,(\boldsymbol{\nu}a)\,\pi \mid \rho\,] \qquad a \notin \mathsf{fn}(\rho) \qquad \pi \ltimes \rho \qquad (\equiv \mathsf{Split}\,\boldsymbol{\nu})$$

$$(\boldsymbol{\nu}a)\,(\pi \mid \mathsf{L}(b)) \equiv (\boldsymbol{\nu}a)\,\pi \mid \mathsf{L}(b) \qquad a \notin \mathsf{fn}(\mathsf{L}(b)) \qquad \pi \propto \mathsf{L}(b) \qquad (\equiv \mathsf{Res}\,\boldsymbol{\nu})$$

We take as $\equiv$ the equivalence relation generated by the above rules and α-conversion. A detail is that $\mathsf{L}(c) = ca \equiv ac = \mathsf{L}(a)$. We also define:

$$\mathsf{conclusion}(a,\pi) \stackrel{\mathsf{def}}{=} \pi \equiv \mathsf{C}[\,\mathsf{L}(a)\,] \wedge a \notin \mathsf{bn}(\pi) \qquad \mathsf{premises}(\mathsf{L}(a)) \stackrel{\mathsf{def}}{=} \mathsf{fn}(\mathsf{L}(a))\backslash a$$

$$\pi \ltimes \rho \stackrel{\mathsf{def}}{=} \mathsf{fn}(\pi) \cap \mathsf{fn}(\rho) = \emptyset \;\vee\; (\,\mathsf{fn}(\pi) \cap \mathsf{fn}(\rho) = a \;\wedge\; \mathsf{conclusion}(a,\pi)\,)$$

$$\pi \propto \mathsf{L}(a) \stackrel{\mathsf{def}}{=} a \notin \mathsf{fn}(\pi) \;\wedge\; \mathsf{L}(a) \notin \star\text{-links} \;\wedge\; b \in \mathsf{premises}(\mathsf{L}(a)) \Rightarrow \mathsf{conclusion}(b,\pi)$$

With $\mathsf{conclusion}(a,\pi)$ we denote that π has a free conclusion a (actually, there could also be another occurrence of a in π, but this is excluded by other conditions in $\equiv$). With $\pi \ltimes \rho$ we denote that either π and ρ are completely independent (i.e., do not communicate, in which case $\mathsf{fn}(\pi) \cap \mathsf{fn}(\rho) = \emptyset$), or that there is a "splitting" cut on some a between π and ρ, or that some $\mathsf{L}(a)$ in π acts as a premise for some link in ρ (in these two cases $\mathsf{fn}(\pi) \cap \mathsf{fn}(\rho) = a$). Hence, when $\pi \ltimes \rho$ holds, π does not depend on ρ. In $\pi \propto \mathsf{L}(a)$, we require that all premises of $\mathsf{L}(a)$ occur in π as conclusions, so π does not depend on $\mathsf{L}(a)$, and that $\mathsf{L}(a)$ is not a $\star$-link. These conditions, and our rules, are heavily inspired by Girard's

empires method [21,22]. We delay the explanation of ($\equiv \mathsf{Split}\,\boldsymbol{\nu}, \equiv \mathsf{Res}\,\boldsymbol{\nu}$) since they are best understood in the context of linear typing.

$$\textit{Typing:}\quad \vdash \pi \triangleright \Gamma \qquad\qquad \textit{Interface:}\quad \Gamma \quad ::= \quad \emptyset \quad | \quad \Gamma, a\colon A \quad | \quad \Gamma, a\colon [A]$$

$$\frac{}{\vdash {\sim}a \triangleright a\colon [A]} \qquad \frac{}{\vdash \boldsymbol{\epsilon} \triangleright \emptyset} \qquad \frac{}{\vdash ab \triangleright a\colon A, b\colon A^{\perp}} \qquad (\mathsf{Open}, \mathsf{Nil}, \mathsf{Ax})$$

$$\frac{\vdash \pi \triangleright \Gamma, a\colon [A] \quad \text{or} \quad \vdash \pi \triangleright \Gamma, a\colon [A^{\perp}]}{\vdash (\boldsymbol{\nu} a\colon A)\,\pi \triangleright \Gamma} \qquad \frac{\vdash \pi \triangleright \Gamma \quad \vdash \rho \triangleright \Delta}{\vdash \pi \mid \rho \triangleright \Gamma, \Delta} \qquad (\mathsf{New}, \mathsf{Mix})$$

$$\frac{\vdash \pi \triangleright \Gamma \quad \sigma_i \text{ is a permutation}}{\vdash \sigma_1(\pi) \triangleright \sigma_2(\Gamma)} \qquad \frac{\vdash \pi \triangleright \Gamma, a\colon A \quad \vdash \rho \triangleright \Delta, a\colon A^{\perp}}{\vdash \pi \mid \rho \triangleright \Gamma, \Delta, a\colon [A]} \qquad (\mathsf{Ex}, \mathsf{Cut})$$

$$\frac{\vdash \pi \triangleright \Gamma, b\colon A \qquad \vdash \rho \triangleright \Delta, c\colon B}{\vdash \overline{a}(b,c) \mid \pi \mid \rho \triangleright \Gamma, \Delta, b\colon [A], c\colon [B], a\colon A \otimes B} \qquad \frac{}{\vdash \overline{a} \triangleright a\colon \mathbf{1}} \qquad (\otimes, \mathbf{1})$$

$$\frac{\vdash \pi \triangleright \Gamma, b\colon A, c\colon B}{\vdash a(b,c) \mid \pi \triangleright \Gamma, b\colon [A], c\colon [B], a\colon A \parr B} \qquad \frac{}{\vdash a \triangleright a\colon \bot} \qquad (\parr, \bot)$$

The typing system ensures correctness of proof structures, i.e., it validates that a process is a proof net. Judgements $\vdash \pi \triangleright \Gamma$ denote that term π can be assigned the *interface* Γ. Also, with Γ, Δ it is always implied that the domains of Γ and Δ must be disjoint. The rules are easy to understand from Linear Logic (with Mix [21]): the interfaces of ($\mathbf{1}, \bot, \otimes, \parr, \mathsf{Cut}, \mathsf{Ax}, \mathsf{Ex}, \mathsf{Mix}$) match the corresponding sequent rules, if we ignore *discharged* occurrences $a\colon [A]$. These denote that "a has been used as A," and are eliminated using (New). Since we don't know whether $[A]$ or $[A^{\perp}]$ was recorded for $(\boldsymbol{\nu} a\colon A)$ (*e.g.*, since cuts are symmetric), in (New) we allow either. The introduction of discharged occurrences takes place through several rules. In (Open), to type the residue ${\sim}a$ of a cut on a. In (Cut), to discharge a which is active. In ($\otimes/\parr$), to discharge a name used in a *suspended cut*, i.e., a wiring between a conclusion and a premise. Finally, in (Ex) we use $\sigma(\pi)$ to denote a permutation of any top-level parallel terms.

We now return to the equivalence rules, noting that they are only sound in the presence of (Mix), by which there is no need for "jump" contexts [22].

($\equiv \mathsf{Split}\,\boldsymbol{\nu}$) The condition $a \notin \mathsf{fn}(\rho)$ ensures that a does not escape its scope, and by $\pi \ltimes \rho$ we have that π does not depend on ρ, therefore ρ may leave the scope $(\boldsymbol{\nu} a)$. For example, $(\boldsymbol{\nu} db)\,(cb \mid ba \mid \overline{y}(a,d) \mid dz) \not\equiv (\boldsymbol{\nu} d)((\boldsymbol{\nu} b)\,(cb \mid ba \mid dz) \mid \overline{y}(a,d))$, since the r.h.s. would be unsound, because $(\boldsymbol{\nu} b)\,(cb \mid ba \mid dz)$ cannot be separated; see type rule ($\otimes$). This is avoided since the separated terms have two free names in common, so $(cb \mid ba \mid dz) \ltimes \overline{y}(a,d)$ does not hold. Similarly, $(\boldsymbol{\nu} db)\,(cb \mid ba \mid \overline{y}(a,d) \mid dz) \not\equiv (\boldsymbol{\nu} d)((\boldsymbol{\nu} b)\,(cb \mid ba \mid \overline{y}(a,d)) \mid dz)$, because the axiom dz, which is needed as premise to $\overline{y}(a,d)$, would be out of context. Again, $(cb \mid ba \mid \overline{y}(a,d)) \ltimes dz$ does not hold, because d appears as premise on the left. Still, we allow: $(\boldsymbol{\nu} db)\,(cb \mid ba \mid \overline{y}(a,d) \mid dz) \equiv (\boldsymbol{\nu} d)\,((\boldsymbol{\nu} b)\,(cb \mid ba) \mid \overline{y}(a,d) \mid dz)$ because dz remains at the same level as $\overline{y}(a,d)$, and $(cb \mid ba) \ltimes (\overline{y}(a,d) \mid dz)$. Note that $(\boldsymbol{\nu} xz)\,(kx \mid xy \mid (\boldsymbol{\nu} a)(ya \mid az) \mid zr) \equiv^3 (\boldsymbol{\nu} xza)\,(kx \mid xy \mid ya \mid az \mid zr)$, although the initial context contains two splitting cuts, on y and z. Thus, our restriction $\pi \ltimes \rho$ is general enough. For the direction of entering the scope, we

must ensure that ρ is not needed outside. For this reason we only apply the rule within some $\mathsf{G}[\,\cdot\,]$, here $(\boldsymbol{\nu} d)\,(\,\cdot\,)$, where the last type rule is unique, so we have an independent sub-proof: nothing necessary can be left out. Without this restriction, $(\boldsymbol{\nu} b)\,(cb \mid ba) \mid dz \mid \overline{y}(a,d) \equiv (\boldsymbol{\nu} b)\,(cb \mid ba \mid dz) \mid \overline{y}(a,d)$, by applying $(\equiv \mathsf{Split}\,\boldsymbol{\nu})$ on $(\boldsymbol{\nu} b)\,(cb \mid ba) \mid dz$, i.e., in $\mathsf{C}[\,\cdot\,] \mid \overline{y}(a,d)$, where $(cb \mid ba) \ltimes dz$ holds. The rule $(\equiv \mathsf{Split}\,\boldsymbol{\nu})$ is extensible with other $\mathsf{G}[\,\cdot\,]$ contexts, introduced later.

$(\equiv \mathsf{Res}\,\boldsymbol{\nu})$ By $\pi \propto \mathsf{L}(b)$, the expelled link is not a $\star$-link: $\otimes$-links split contexts; for units, $(\boldsymbol{\nu} a)(ea \mid ad) \mid \overline{c}(d,b) \mid \overline{b} \not\equiv (\boldsymbol{\nu} a)(ea \mid ad \mid \overline{b}) \mid \overline{c}(d,b)$, but if $\overline{b} \notin \star$-links, $(da \mid ae) \propto \overline{b}$. Also by $\pi \propto \mathsf{L}(b)$, $b \notin \mathsf{fn}(\pi)$, for example $(\boldsymbol{\nu} a)\,(ea \mid ab \mid cd \mid bc) \not\equiv (\boldsymbol{\nu} a)\,(ea \mid ab \mid cd) \mid bc$, since the r.h.s. cannot be typed (bc was "linking" two cuts in the scope). On the other hand, $(\boldsymbol{\nu} a)\,(ea \mid ab \mid cd \mid bc) \equiv (\boldsymbol{\nu} a)\,(ea \mid ab \mid bc) \mid cd$ is sound. If $b \in \mathsf{fn}(\pi)$, then under appropriate contexts we may apply $(\equiv \mathsf{Split}\,\boldsymbol{\nu})$. Typically, scopes "swallow" unprotected links such as $(\parr)$ when both premises are inside, the case of just one premise in the scope is handled by $(\equiv \mathsf{Split}\,\boldsymbol{\nu})$, and we forbid the case where both premises are outside. For example, with both premises x, y, of $b(x,y)$ in $(\boldsymbol{\nu} a)$: $(\boldsymbol{\nu} a)\,(xa \mid ay) \mid b(x,y) \equiv (\boldsymbol{\nu} a)\,(xa \mid ay \mid b(x,y))$; with only x in $(\boldsymbol{\nu} a)$: $(\boldsymbol{\nu} a)\,(xa \mid az) \mid b(x,y) \mid yc \not\equiv (\boldsymbol{\nu} a)\,(xa \mid az \mid b(x,y)) \mid yc$; with both x, y, outside of $(\boldsymbol{\nu} a)$: $(\boldsymbol{\nu} a)\,\pi \mid b(x,y) \mid xy) \not\equiv (\boldsymbol{\nu} a)\,(\pi \mid b(x,y)) \mid xy$. In the last two cases, the r.h.s. is untypable even if the l.h.s. was well-typed. However, using $(\equiv \mathsf{Split}\,\boldsymbol{\nu})$ we allow $(\boldsymbol{\nu} x)\,((\boldsymbol{\nu} a)\,(xa \mid az) \mid b(x,y) \mid yc) \equiv (\boldsymbol{\nu} xa)\,(xa \mid az \mid b(x,y) \mid yc)$.

As parallel composition is n-ary, to match linear sequents, we shall not worry with associativity which in general does not hold. It can be recovered with $\mathsf{G}[\,(\pi_1 \mid \pi_2) \mid \pi_3\,] \equiv \mathsf{G}[\,\pi_1 \mid \pi_2 \mid \pi_3\,] \equiv \mathsf{G}[\,\pi_1 \mid (\pi_2 \mid \pi_3)\,]$ whenever $(\pi_1 \mid \pi_2) \ltimes \pi_3$ and $(\pi_2 \mid \pi_3) \ltimes \pi_1$, taking $\mathsf{G}[\,\cdot\,] \stackrel{\mathsf{def}}{=} \cdots \mid (\,\cdot\,)$, i.e., with parentheses in the syntax.

Although there are no prefixes, we can obtain a familiar π-calculus notation:

$$a(x)\cdot\pi \stackrel{\mathsf{def}}{=} (\boldsymbol{\nu} x)\,(ax \mid \pi) \qquad a(x,y)\cdot\pi \stackrel{\mathsf{def}}{=} (\boldsymbol{\nu} xy)\,(a(x,y) \mid \pi)$$

We omitted $\overline{a}(x,y)\cdot\pi$ which similar to $a(x,y)\cdot\pi$. Observe that $\pi\{{}^{a}\!/\!_{x}\} = a(x)\cdot\pi$, thus: $(\pi\{{}^{b}\!/\!_{c}\})\{{}^{a}\!/\!_{b}\} \rightsquigarrow^2 \pi\{{}^{a}\!/\!_{c}\} \,{}^2\!\!\leftsquigarrow (\boldsymbol{\nu} b)\,(ba \mid b(c)\cdot\pi)$. Using (New) and $(\mathsf{Ax}, \parr, \otimes)$:

$$\frac{\vdash \pi \triangleright \Gamma, x\colon A}{\vdash a(x)\cdot\pi \triangleright \Gamma, a\colon A} \qquad \frac{\vdash \pi \triangleright \Gamma, x\colon A, y\colon B}{\vdash a(x,y)\cdot\pi \triangleright \Gamma, a\colon A \parr B} \qquad \frac{\vdash \pi \triangleright \Gamma, x\colon A \quad \rho \triangleright \Delta, y\colon B}{\vdash \overline{a}(x,y)\cdot(\pi \mid \rho) \triangleright \Gamma, \Delta, a\colon A \otimes B}$$

Abramsky's cut rule (see [9]) is similarly derived with an application of (New) under a (Cut). We can thus encode a linear π-calculus with internal mobility (the Private π-calculus [38]), enjoying the deadlock-freedom of proof nets.

It is worth noting that the standard presentation of typed π-calculus (including *sessions* [39,26]) employs a rule similar to $(\parr)$ for both input and output, so it allows to type: $(\boldsymbol{\nu} a)\,(a(x,y).\overline{x}\langle y\rangle \mid \overline{a}\langle b,c\rangle \mid \overline{c}\langle b\rangle) \to \overline{b}\langle c\rangle \mid \overline{c}\langle b\rangle$, which is undesirable if b and c are linear, i.e., $\overline{b}\langle c\rangle \mid \overline{c}\langle b\rangle \cong \mathbf{0}$ but the term's derivation is not "cut-free." In untyped δ-calculus this translates to: $(\boldsymbol{\nu} a)\,(a(x,y)\cdot xy \mid \overline{a}(b,c) \mid cb) \rightsquigarrow^* cc \mid {\sim}b$, where cc is what is known as a *vicious circle* in proof nets [22]. Type rule $(\otimes)$ rejects $\overline{a}(b,c) \mid cb$, and such problematic situations never take place.

The Additive Fragment The extension to Multiplicative Additive Linear Logic (MALL), which allows to define conditionals, is obtained by adding the following productions. We ignore the units ($\top/\mathbf{0}$) which are of little interest [22].

$\mathsf{L}(a) ::=$ *as before, adding:* $\quad \overline{a}_{\mathbf{l}}\langle b\rangle$ (plus left)

$\quad a_{\mathbf{l}}(\boldsymbol{\nu} c\colon A)\boldsymbol{.}\pi + a_{\mathbf{r}}(\boldsymbol{\nu} d\colon B)\boldsymbol{.}\rho$ (with $\star$) $\quad \overline{a}_{\mathbf{r}}\langle b\rangle$ (plus right)

Summations are *protected*: $(\boldsymbol{\nu} a)\,(ea \mid ab \mid z_{\mathbf{l}}(\boldsymbol{\nu} x)\boldsymbol{.}(bc \mid \overline{x}) + z_{\mathbf{r}}(\boldsymbol{\nu} y)\boldsymbol{.}(bc \mid \overline{y}) \mid cd) \not\equiv (\boldsymbol{\nu} a)\,(ea \mid ab \mid cd) \mid z_{\mathbf{l}}(\boldsymbol{\nu} x)\boldsymbol{.}(bc \mid \overline{x}) + z_{\mathbf{r}}(\boldsymbol{\nu} y)\boldsymbol{.}(bc \mid \overline{y})$, since bc must not leave the scope.

$$a_{\mathbf{l}}(\boldsymbol{\nu} c)\boldsymbol{.}\pi + a_{\mathbf{r}}(\boldsymbol{\nu} d)\boldsymbol{.}\rho \mid \overline{a}_{\mathbf{l}}\langle b\rangle \overset{a}{\rightsquigarrow} \pi\{{}^{b}\!/_{c}\} \qquad (\& - \oplus_1)$$

$$a_{\mathbf{l}}(\boldsymbol{\nu} c)\boldsymbol{.}\pi + a_{\mathbf{r}}(\boldsymbol{\nu} d)\boldsymbol{.}\rho \mid \overline{a}_{\mathbf{r}}\langle b\rangle \overset{a}{\rightsquigarrow} \rho\{{}^{b}\!/_{d}\} \qquad (\& - \oplus_2)$$

$$\mathsf{G}[\cdot] \;::=\; \cdots \;\mid\; a_{\mathbf{l}}(\boldsymbol{\nu} c)\boldsymbol{.}\mathsf{G}[\cdot] + a_{\mathbf{r}}(\boldsymbol{\nu} d)\boldsymbol{.}\pi \;\mid\; a_{\mathbf{l}}(\boldsymbol{\nu} c)\boldsymbol{.}\pi + a_{\mathbf{r}}(\boldsymbol{\nu} d)\boldsymbol{.}\mathsf{G}[\cdot]$$

With additives there is a phenomenon of superposition, as exactly one of the sides will be chosen, left with $\overline{a}_{\mathbf{l}}\langle b\rangle$ or right with $\overline{a}_{\mathbf{r}}\langle b\rangle$. An example of the left case is: $\overline{a}_{\mathbf{l}}\langle b\rangle \mid a_{\mathbf{l}}(\boldsymbol{\nu} c)\boldsymbol{.}cz + a_{\mathbf{r}}(\boldsymbol{\nu} d)\boldsymbol{.}\rho \overset{a}{\rightsquigarrow} cz\{{}^{b}\!/_{c}\} \rightsquigarrow^{2} {\sim}a \mid bz$. Both summands are prefixed (proof theoretically: boxed), marking a moment of sequentiality, or *laziness*. Once a side is chosen, applying $(\oplus_1 - \&)$, the bound name c is *extruded* and wired to b, becoming a scope $(\boldsymbol{\nu} c)\,(bc \mid cz)$ which is abbreviated as before to $cz\{{}^{b}\!/_{c}\}$. The other side, $a_{\mathbf{r}}(\boldsymbol{\nu} d)\boldsymbol{.}\rho$, vanishes, which is why additives are the natural choice for *if-then-else* constructs, or *objects* as summations of *methods*.

$$(\boldsymbol{\nu} b)\,(a_{\mathbf{l}}(\boldsymbol{\nu} x)\boldsymbol{.}\pi_1 + a_{\mathbf{r}}(\boldsymbol{\nu} y)\boldsymbol{.}\pi_2) \equiv a_{\mathbf{l}}(\boldsymbol{\nu} x)\boldsymbol{.}(\boldsymbol{\nu} b)\,\pi_1 + a_{\mathbf{r}}(\boldsymbol{\nu} y)\boldsymbol{.}(\boldsymbol{\nu} b)\,\pi_2 \qquad (\equiv \boldsymbol{\nu}\&)$$

$$a_{\mathbf{l}}(\boldsymbol{\nu} x)\boldsymbol{.}(b_{\mathbf{l}}(\boldsymbol{\nu} c)\boldsymbol{.}\pi_1 + b_{\mathbf{r}}(\boldsymbol{\nu} d)\boldsymbol{.}\pi_2) + a_{\mathbf{r}}(\boldsymbol{\nu} y)\boldsymbol{.}(b_{\mathbf{l}}(\boldsymbol{\nu} c)\boldsymbol{.}\pi_3 + b_{\mathbf{r}}(\boldsymbol{\nu} d)\boldsymbol{.}\pi_4) \qquad (\equiv \&\&)$$
$$\equiv b_{\mathbf{l}}(\boldsymbol{\nu} c)\boldsymbol{.}(a_{\mathbf{l}}(\boldsymbol{\nu} x)\boldsymbol{.}\pi_1 + a_{\mathbf{r}}(\boldsymbol{\nu} y)\boldsymbol{.}\pi_3) + b_{\mathbf{r}}(\boldsymbol{\nu} d)\boldsymbol{.}(a_{\mathbf{l}}(\boldsymbol{\nu} x)\boldsymbol{.}\pi_2 + a_{\mathbf{r}}(\boldsymbol{\nu} y)\boldsymbol{.}\pi_4)$$

$$\mathsf{G}[\; a_{\mathbf{l}}(\boldsymbol{\nu} x)\boldsymbol{.}(\pi_1 \mid \rho) + a_{\mathbf{r}}(\boldsymbol{\nu} y)\boldsymbol{.}(\pi_2 \mid \rho)\;] \qquad (\equiv \mathsf{Split}\,\&)$$
$$\equiv \mathsf{G}[\; a_{\mathbf{l}}(\boldsymbol{\nu} x)\boldsymbol{.}\pi_1 + a_{\mathbf{r}}(\boldsymbol{\nu} y)\boldsymbol{.}\pi_2 \mid \rho\;] \qquad a, x, y \notin \mathsf{fn}(\rho) \qquad \pi_i \ltimes \rho$$

$$a_{\mathbf{l}}(\boldsymbol{\nu} x)\boldsymbol{.}(\pi_1 \mid \mathsf{L}(b)) + a_{\mathbf{r}}(\boldsymbol{\nu} y)\boldsymbol{.}(\pi_2 \mid \mathsf{L}(b)) \qquad (\equiv \mathsf{Res}\,\&)$$
$$\equiv a_{\mathbf{l}}(\boldsymbol{\nu} x)\boldsymbol{.}\pi_1 + a_{\mathbf{r}}(\boldsymbol{\nu} y)\boldsymbol{.}\pi_2 \mid \mathsf{L}(b) \qquad a, x, y \notin \mathsf{fn}(\mathsf{L}(b)) \qquad \pi_i \propto \mathsf{L}(b)$$

As before, the new contexts $\mathsf{G}[\cdot]$ define complete sub-proofs, facilitating safe commutations; see (&). With $(\equiv \&\&)$ we capture the commutation (&)-(&) of Linear Logic. Using $(\equiv \mathsf{Split}\,\&)$ with $\mathsf{G}[\cdot] = (\boldsymbol{\nu} c)(\cdot)$ and $\rho = yc$, then $(\equiv \boldsymbol{\nu}\&)$:

$$(a_{\mathbf{l}}(\boldsymbol{\nu} b)\boldsymbol{.}bc + a_{\mathbf{r}}(\boldsymbol{\nu} d)\boldsymbol{.}dc)\{{}^{y}\!/_{c}\} \equiv a_{\mathbf{l}}(\boldsymbol{\nu} b)\boldsymbol{.}bc\{{}^{y}\!/_{c}\} + a_{\mathbf{r}}(\boldsymbol{\nu} d)\boldsymbol{.}dc\{{}^{y}\!/_{c}\} \rightsquigarrow^{4} a_{\mathbf{l}}(\boldsymbol{\nu} b)\boldsymbol{.}by + a_{\mathbf{r}}(\boldsymbol{\nu} d)\boldsymbol{.}dy$$

For correctness, anything that enters these boxes is duplicated, which is not efficient but it's needed for cut-elimination, even if in a practical language it would be applied selectively. A possible utility is for *speculative execution*, by reducing under both terms of a summation using terms from the outside, even if half will eventually be thrown away. This is not unreasonable in a parallel world. Another feasible optimisation is to reduce all multiplicative cuts during compilation, eliminating some of the the indirections that arise from linking, since the $\otimes/\parr$ fragment normalises in a polynomial number of steps [21].

The additional typing rules match the corresponding logical rules:

$$\frac{\vdash \pi \triangleright \Gamma, b\colon A \qquad \vdash \rho \triangleright \Gamma, d\colon B}{\vdash a_{\mathbf{l}}(\boldsymbol{\nu} b\colon A)\boldsymbol{.}\pi + a_{\mathbf{r}}(\boldsymbol{\nu} d\colon B)\boldsymbol{.}\rho \triangleright \Gamma, a\colon A \,\&\, B} \qquad (\&)$$

$$\frac{\vdash \pi \triangleright \Gamma, b\colon A}{\vdash \overline{a}_{\mathbf{l}}\langle b\rangle \mid \pi \triangleright \Gamma, b\colon [A], a\colon A \oplus B} \qquad \frac{\vdash \pi \triangleright \Gamma, c\colon B}{\vdash \overline{a}_{\mathbf{r}}\langle c\rangle \mid \pi \triangleright \Gamma, c\colon [B], a\colon A \oplus B} \qquad (\oplus_{1,2})$$

The Exponential Fragment The exponential links are responsible for the expressive power of Linear Logic and embody *stored* terms, $!\overline{a}(\boldsymbol{\nu} c).\pi$, and the operations of dereliction (to *use* or *dereference*) $?a(b)$, contraction (to *copy*) $?a(b \otimes c)$, and weakening (to *delete*) $?a(*)$. Stored terms resemble π-calculus replication, and in fact inspired the familiar notation $!P$ [32].

$\mathsf{L}(a) ::=$		*as before, adding:*	
$!\overline{a}(\boldsymbol{\nu} c\colon A).\pi$	(stored term $\star$)	$?a(b \otimes c)$	(contraction)
$?a(b)$	(dereliction)	$?a(*)$	(weakening $\star$)

We extend the reductions writing $\overrightarrow{?e(*)}$ to mean $?e_1(*) \mid \cdots \mid ?e_n(*)$, similarly for $?\overrightarrow{e(x \otimes y)}$, and $c\vec{e}$ for the set $\{c, e_1, \ldots, e_n\}$. Taking $\mathsf{fn}(\pi) = c\vec{e}$:

$$?a(b) \mid !\overline{a}(\boldsymbol{\nu} c).\pi \overset{a}{\rightsquigarrow} \pi\{{}^{b}/_{c}\} \qquad (?\mathsf{D} - !) \qquad \mathsf{G}[\cdot] ::= \cdots \mid !\overline{a}(\boldsymbol{\nu} c).\mathsf{G}[\cdot]$$

$$?a(*) \mid !\overline{a}(\boldsymbol{\nu} c).\pi \overset{a}{\rightsquigarrow} \overrightarrow{?e(*)} \qquad (?\mathsf{W} - !) \qquad ?a(b \otimes c) \mid ?b(*) \overset{b}{\rightsquigarrow} ac \qquad (?\mathsf{C} - ?\mathsf{W}\ [28])$$

$$(\boldsymbol{\nu}\vec{e}\colon \vec{A})\,(\,?a(b \otimes d) \mid !\overline{a}(\boldsymbol{\nu} c).\pi \mid \rho\,) \qquad (?\mathsf{C} - !)$$

$$\overset{a}{\rightsquigarrow} (\boldsymbol{\nu}\vec{e}\colon \vec{A})(\boldsymbol{\nu}\vec{x}\colon \vec{A})(\boldsymbol{\nu}\vec{y}\colon \vec{A})\left(?\overrightarrow{e(x \otimes y)} \;\middle|\; !\overline{b}(\boldsymbol{\nu} c).\pi\{{}^{\vec{x}}/_{\vec{e}}\} \;\middle|\; !\overline{d}(\boldsymbol{\nu} c).\pi\{{}^{\vec{y}}/_{\vec{e}}\} \;\middle|\; \rho\right)$$

Using dereliction, $(?\mathsf{D} - !)$, we have: $?a(b) \mid !\overline{a}(\boldsymbol{\nu} c).\pi \rightsquigarrow \sim a \mid (\boldsymbol{\nu} c)\,(bc \mid \pi)$. The stored term opens into a scope, *extruding* $(\boldsymbol{\nu} c)$ to encompass b, so that bc can be wired, providing access to π; thus the exponential *becomes* linear. Now consider a weakening reduction, using $(?\mathsf{W} - !)$:

$$?a(*) \mid !\overline{a}(\boldsymbol{\nu} b).\Big(?c(*) \mid (\boldsymbol{\nu} x)\,(?d(x) \mid \overline{x})\Big) \mid !\overline{c}\ldots \mid !\overline{d}\ldots \rightsquigarrow \sim a \mid ?c(*) \mid ?d(*) \mid !\overline{c}\ldots \mid !\overline{d}\ldots$$

In order to preserve linearity, i.e., that names are used exactly once, $!\overline{c}\ldots$ and $!\overline{d}\ldots$ must be also deleted. When there are no dependencies, i.e., free names, the term is simply deleted: $?a(*) \mid !\overline{a}(\boldsymbol{\nu} x).\overline{x} \rightsquigarrow \sim a$. The rule $(?\mathsf{C} - ?\mathsf{W})$ is by Kesner and Lengrand [28], and constitutes a nice optimisation: if a is copied as b and c, but b is to be deleted, we can avoid the potentially expensive copy and directly link a and c. By $\equiv$, the same can be done for the symmetric case of c deleted.

Copying is implemented with $(?\mathsf{C} - !)$: from $!\overline{a}(\boldsymbol{\nu} c).\pi$ we obtain $!\overline{b}(\boldsymbol{\nu} c).\pi\{{}^{\vec{x}}/_{\vec{e}}\}$ and $!\overline{d}(\boldsymbol{\nu} c).\pi\{{}^{\vec{y}}/_{\vec{e}}\}$. For each of the free names $\vec{e}$ of π, we add contractions $?e_1(x_1 \otimes y_1) \mid \cdots \mid ?e_n(x_n \otimes y_n)$, with each x_i appearing in $!\overline{b}\ldots$, and each y_i in $!\overline{d}\ldots$, since the same e_i cannot appear in both. We resort to a contextual rule, since $(\boldsymbol{\nu}\vec{e}\colon \vec{A})$ recovers the types for $\vec{x}$ and $\vec{y}$. This is also necessary in an interpretation without (Mix); otherwise, in a type-free reduction the context can be removed.

$$(\boldsymbol{\nu} x)\,(!\overline{a}(\boldsymbol{\nu} b).\pi \mid !\overline{x}(\boldsymbol{\nu} d).\rho) \equiv !\overline{a}(\boldsymbol{\nu} b).(\boldsymbol{\nu} x)(\pi \mid !\overline{x}(\boldsymbol{\nu} d).\rho) \qquad (\equiv \boldsymbol{\nu}!)$$
$$b \notin \mathsf{fn}(\rho) \qquad \mathsf{conclusion}(x, \pi)$$

$$(\boldsymbol{\nu} y)\,(!\overline{a}(\boldsymbol{\nu} b).\pi \mid ?x(y)) \equiv !\overline{a}(\boldsymbol{\nu} b).(\boldsymbol{\nu} y)(\pi \mid ?x(y)) \quad \pi \equiv \mathsf{C}[\,?y(-)\,] \quad y \notin \mathsf{bn}(\pi) \quad (\equiv ?\mathsf{D}!)$$

$$(\boldsymbol{\nu} yz)\,(!\overline{a}(\boldsymbol{\nu} b).\pi \mid ?x(y \otimes z)) \equiv !\overline{a}(\boldsymbol{\nu} b).(\boldsymbol{\nu} yz)(\pi \mid ?x(y \otimes z)) \qquad (\equiv ?\mathsf{C}!)$$

$$\mathsf{G}[\,!\overline{a}(\boldsymbol{\nu} b).\pi \mid ?x(*)\,] \equiv \mathsf{G}[\,!\overline{a}(\boldsymbol{\nu} b).(\pi \mid ?x(*))\,] \qquad (\equiv ?\mathsf{W}!)$$

$$?x(y \otimes z) \equiv ?x(z \otimes y) \qquad ?a(b \otimes c) \mid ?b(d \otimes e) \equiv ?a(b \otimes e) \mid ?b(d \otimes c) \qquad (\equiv ?\mathsf{CM}, \equiv ?\mathsf{AS})$$

The rules $(\equiv ?\mathsf{C}!, \equiv ?\mathsf{CM}, \equiv ?\mathsf{AS})$ are adapted from the *AC Congruence* of Di Cosmo and Guerrini [13]: $(\equiv ?\mathsf{C}!)$ captures the commutation of contraction and

promotion, and ($\equiv$?CM, $\equiv$?AS) turn contraction into a commutative-associative operation. The right-to-left direction of ($\equiv$?W!) is based on a contraction rule by Kesner and Lengrand [28], but here we make it symmetric and also place it under a context G which guarantees that $?x(*)$ is not needed outside the box, *e.g.*, as premise to some link; ($\equiv$?C!) needs ($\boldsymbol{\nu} yz$), from which we also have $y, z \in \mathsf{fn}(\pi)$, and similarly for y in ($\equiv$?D!). Note that in ($\equiv$?D!, $\equiv$?C!, $\equiv$?W!), we have $x \notin \mathsf{fn}(\pi)$ by typing: on the left the term would be untypable, and on the right some $x\colon [A]$ would appear in the interface, which is forbidden in (P). Rule ($\equiv$?D!) is our contribution, and allows a dereliction to cross a box boundary as long as its premise is also a ?-link; the condition $y \notin \mathsf{bn}(\pi)$ is needed as we do not assume the variable convention. It is easy to see why this rule is sound, as in this special case promotion (P) and dereliction (D) commute:

$$?x(y) \mid ?y(-) \quad \text{induces} \quad x\colon ??B \qquad \dfrac{\dfrac{\vdash ?\Gamma, ?B, A}{\vdash ?\Gamma, ??B, A}\,(\mathsf{D})}{\vdash ?\Gamma, ??B, !A}\,(\mathsf{P}) \qquad \dfrac{\dfrac{\vdash ?\Gamma, ?B, A}{\vdash ?\Gamma, ?B, !A}\,(\mathsf{P})}{\vdash ?\Gamma, ??B, !A}\,(\mathsf{D})$$

Above, $?y(-)$ stands for any ?-link. The equivalence ($\equiv \boldsymbol{\nu}!$), whose left-to-right direction is standard from proof nets [21], brings one box into another, enabling a cut on x even if $!\overline{a}\ldots$ is *never* opened. If x appears in π as premise then the right-to-left direction is unsound, so we require that x is a conclusion in π.

$$\dfrac{\vdash \pi \triangleright ?\Gamma, b\colon A}{\vdash !\overline{a}(\boldsymbol{\nu} b\colon A).\pi \triangleright ?\Gamma, a\colon !A} \qquad \dfrac{\vdash \pi \triangleright \Gamma, b\colon A}{\vdash ?a(b) \mid \pi \triangleright \Gamma, b\colon [A], a\colon ?A} \qquad (\mathsf{P},\ \mathsf{D})$$

$$\dfrac{\vdash \pi \triangleright \Gamma, b\colon ?A, c\colon ?A}{\vdash ?a(b \mathbin{⅋} c) \mid \pi \triangleright \Gamma, b\colon [?A], c\colon [?A], a\colon ?A} \qquad \dfrac{}{\vdash ?a(*) \triangleright a\colon ?A} \qquad (\mathsf{C},\ \mathsf{W})$$

When typing a term that can be copied (resp. deleted) using the promotion rule (P), it must only refer to other such terms, which are copied (resp. deleted) accordingly during reduction. This is ensured by typing under $?\Gamma$, meaning that all elements of Γ are of the form $a\colon ?A$. The weakening rule (W) needs no context as it has no premise and by the use of (Mix) it needs no default jump [22]. Stored terms are protected: *e.g.*, in $(\boldsymbol{\nu} a)(ra \mid ab \mid !\overline{e}(\boldsymbol{\nu} z).(\boldsymbol{\nu} xy)(z(x,y) \mid xb \mid yc) \mid cd) \not\equiv (\boldsymbol{\nu} a)(ra \mid ab \mid cd) \mid !\overline{e}(\boldsymbol{\nu} z).(\boldsymbol{\nu} xy)(z(x,y) \mid xb \mid yc)$ the r.h.s. is incorrect. Also, $?a(*)$ is protected, like the units, because it has no premise to guide $\equiv$.

The Second-Order Fragment Second-order features are responsible for considerable expressive power, and (in combination with exponentials) allow to define polymorphic data types and primitive recursion.

$\pi, \rho ::=$	*as before, adding:*	$\mathsf{L}(a) ::=$	*as before, adding:*
$(\boldsymbol{\nu}\mathsf{X})\,\pi$	(type variable scope)	$a[\mathsf{X}, b]$	(forall)
$\mathsf{X}{=}B$	(type alias)	$\overline{a}[B, b]$	(exists)

The logical connectives are forall: $a[\mathsf{X}, b]$ and exists: $\overline{a}[B, b]$. Type substitution is defined in the usual way: $\mathrm{A}[B/\mathsf{X}]$ stands for A with B for X, and $B^{\perp}$ for $\mathsf{X}^{\perp}$ [21],

and can also be applied to terms with $\pi[B/\mathsf{X}]$, substituting all free occurrences of X in type annotations and aliases. The type variable X is bound in $(\boldsymbol{\nu}\mathsf{X})\,\pi$.

$$a[\mathsf{X},b] \mid \overline{a}[B,c] \overset{a}{\rightsquigarrow} \mathsf{X}{=}B \mid bc \qquad (\forall - \exists) \qquad \mathsf{G}[\,\cdot\,] \;::=\; \cdots \;\mid\; (\boldsymbol{\nu}\mathsf{X})\,\mathsf{G}[\,\cdot\,]$$
$$(\boldsymbol{\nu}\mathsf{X})\,(\mathsf{X}{=}B \mid \pi) \rightsquigarrow \pi[B/\mathsf{X}] \qquad (\sim\mathsf{X}{=})$$

The reduction $(\forall - \exists)$ releases a *type alias* $\mathsf{X}{=}B$, which may cross the boundary of names by $\equiv$. Eventually, $\mathsf{X}{=}B$ may reach the binder of X and disappear with $(\sim\mathsf{X}{=})$, *effecting* the type substitution. There is an alternative, very eligible possibility for this fragment. Forget about type aliases $\mathsf{X}{=}B$ and the rule $(\sim\mathsf{X}{=})$, and define $(\forall - \exists)\prime$ as: $(\boldsymbol{\nu}\mathsf{X}a)\,(\mathsf{C}[\,a[\mathsf{X},b] \mid \overline{a}[B,c]\,]) \rightsquigarrow \mathsf{C}[\,bc\,][B/\mathsf{X}]$, assuming a is not bound in C. Like $(\sim\mathsf{X}{=})$, $(\forall - \exists)\prime$ can increase the size of a term drastically, since B can substitute multiple occurrences of X, and it may contain free type variables which proliferate and cause size explosion after iterated substitutions. This is undesirable, especially in *light* variations of proof nets (see [6]) where reduction must respect tight bounds. However, unlike $(\forall - \exists)\prime$, the rule $(\sim\mathsf{X}{=})$ can be delayed, or even ignored (reduction modulo), because the alias is enough for typing. In implementations the types may be erased, which works fine: $\mathsf{erase}(a[\mathsf{X},b])) = \mathsf{erase}(\overline{a}[B,b]) = ab$. In that case, instances of $(\forall - \exists)$ become axiom cuts. The presence of bound type variables induces new equivalences:

$$(\boldsymbol{\nu}\mathsf{X})(\boldsymbol{\nu}\mathsf{Y})\,\pi \equiv (\boldsymbol{\nu}\mathsf{Y})(\boldsymbol{\nu}\mathsf{X})\,\pi \qquad (\equiv \mathsf{XX})$$
$$(\boldsymbol{\nu}a\colon A)(\boldsymbol{\nu}\mathsf{X})\,\pi \equiv (\boldsymbol{\nu}\mathsf{X})(\boldsymbol{\nu}a\colon A)\,\pi \qquad \mathsf{X} \notin \mathsf{ftv}(A) \qquad (\equiv \boldsymbol{\nu}\mathsf{X})$$
$$(\boldsymbol{\nu}\mathsf{X})\,(a_{\mathbf{l}}(\boldsymbol{\nu}x\colon A).\pi_1 + a_{\mathbf{r}}(\boldsymbol{\nu}y\colon B).\pi_2) \qquad (\equiv \mathsf{X}\&)$$
$$\equiv a_{\mathbf{l}}(\boldsymbol{\nu}x\colon A).(\boldsymbol{\nu}\mathsf{X})\,\pi_1 + a_{\mathbf{r}}(\boldsymbol{\nu}y\colon B).(\boldsymbol{\nu}\mathsf{X})\,\pi_2 \qquad \mathsf{X} \notin \mathsf{ftv}(A,B)$$
$$\mathsf{G}[\,(\boldsymbol{\nu}\mathsf{X})\,(\pi \mid \rho)\,] \equiv \mathsf{G}[\,(\boldsymbol{\nu}\mathsf{X})\,\pi \mid \rho\,] \qquad \mathsf{X} \notin \mathsf{ftv}(\rho) \qquad \pi \ltimes \rho \qquad (\equiv \mathsf{Split}\,\mathsf{X})$$
$$(\boldsymbol{\nu}\mathsf{X})\,(\pi \mid \mathsf{L}(b)) \equiv (\boldsymbol{\nu}\mathsf{X})\,\pi \mid \mathsf{L}(b) \qquad \mathsf{X} \notin \mathsf{ftv}(\mathsf{L}(b)) \qquad \pi \propto \mathsf{L}(b) \qquad (\equiv \mathsf{Res}\,\mathsf{X})$$

Typing is extended with $(\forall, \exists)$ rules and the bookkeeping (NewX, SubX1, SubX2).

$$\frac{\vdash \pi \triangleright \Gamma, b\colon A}{\vdash a[\mathsf{X},b] \mid \pi \triangleright \Gamma, \mathsf{X}, b\colon [A], a\colon \forall\mathsf{X}.A} \qquad \frac{\vdash \pi \triangleright \Gamma, b\colon A[B/\mathsf{X}]}{\vdash \overline{a}[B,b] \mid \pi \triangleright \Gamma, b\colon [A[B/\mathsf{X}]], a\colon \exists\mathsf{X}.A} \quad (\forall,\ \exists)$$

$$\frac{\vdash \pi \triangleright \Gamma, \mathsf{X} \quad \mathsf{X} \notin \mathsf{ftv}(\Gamma)}{\vdash (\boldsymbol{\nu}\mathsf{X})\,\pi \triangleright \Gamma} \qquad \frac{\vdash \pi[B/\mathsf{X}] \triangleright \Gamma[B/\mathsf{X}]}{\vdash \mathsf{X}{=}B \mid \pi \triangleright \Gamma, \mathsf{X}} \qquad \frac{}{\vdash \mathsf{X}{=}B \triangleright \mathsf{X}} \quad (\mathsf{NewX},\ \mathsf{SubX1/2})$$

(SubX2) is needed for $(\boldsymbol{\nu}\mathsf{X})\mathsf{X}{=}B$, and (SubX1) is for when a type has been instantiated with an alias, but no substitution has taken place, as below:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash bc \triangleright b\colon B^{\perp}, c\colon B}\,(\mathsf{Ax})}{\vdash a(b,c) \mid bc \triangleright a\colon B^{\perp} \parr B, b\colon [B^{\perp}], c\colon [B]}\,(\parr)}{\vdash \mathsf{X}{=}B \mid a(b,c) \mid bc \triangleright a\colon B^{\perp} \parr B, b\colon [\mathsf{X}^{\perp}], c\colon [\mathsf{X}], \mathsf{X}}\,(\mathsf{SubX2})}{\vdash (\boldsymbol{\nu}\mathsf{X})(\boldsymbol{\nu}b\colon \mathsf{X})(\boldsymbol{\nu}c\colon \mathsf{X}^{\perp})\,(\mathsf{X}{=}B \mid a(b,c) \mid bc) \triangleright a\colon B^{\perp} \parr B}\,(\mathsf{New}, \mathsf{NewX})$$

The above is in fact a reduct of the translation $[\![(\Lambda X.\lambda x^{\mathsf{X}}.x)\,B]\!]_a$; see Section 5. Similarly, we can obtain $\vdash (\boldsymbol{\nu}\mathsf{X})(\,(\boldsymbol{\nu}b\colon \mathsf{X})(\boldsymbol{\nu}c\colon \mathsf{X}^{\perp})\,(a(b,c) \mid bc) \mid \mathsf{X}{=}B\,) \triangleright a\colon B^{\perp} \parr B$.

The Interpretation of Affine Logic The &-links induce boxes (i.e., prefixes), which are useful as a *lazy* feature but ultimately impede parallelism. Proof-theoretically, this constitutes a defect of proof nets, which are meant to abstract the bureaucratic commutations that belong in the sequent world, however the existing solutions for removing boxes [22,27], albeit ingenious, are too complicated for our purposes. Affine Linear Logic [6] solves this problem by cutting the knot: it allows unrestricted weakening, so *anything* can be discarded, which leads to multiplicative encodings of &-links since it now holds that $\vdash (A \otimes B) \multimap (A \,\&\, B)$. We add a *deletion* $\star$-link $a(*)$ (cf. weakening link of Affine proof nets [6]; ϵ-link of Interaction Nets [29]), typed with (AW). (W) and $?a(*)$ are removed.

$$\frac{}{\vdash a(*) \triangleright a\colon A}\ (\mathsf{AW}) \qquad a(*) \mid \mathsf{L}(a) \overset{a}{\rightsquigarrow} \overrightarrow{b(*)} \qquad \mathsf{fn}(\mathsf{L}(a)) = a\vec{b} \qquad (\mathsf{AW}-\mathsf{Link})$$

$$\frac{\vdash \pi \triangleright \Gamma}{\vdash \pi \triangleright \Gamma, \mathsf{X}}\ (\mathsf{XW}) \qquad (\boldsymbol{\nu}\mathsf{X})\,\pi \rightsquigarrow \pi \qquad \mathsf{X} \notin \mathsf{ftv}(\pi) \qquad (\sim\mathsf{AX}{=})$$

Deletion eliminates anything it cuts with, and propagates, preserving typability. For example, $a(*) \mid a(*) \rightsquigarrow \sim a$. As another example, $a(*) \mid \overline{a}(b,c) \mid \pi \mid \rho \rightsquigarrow b(*) \mid c(*) \mid \sim a \mid \pi \mid \rho$. Assume that $\overline{a}(b,c)$ is typed with π in the left premise and ρ in the right; then the contractum is typed with (Open), (Mix), and two instances of (Cut) for $\pi \mid b(*)$ and $\rho \mid c(*)$, respectively.

Conditionals can be encoded reasonably with multiplicatives, *e.g.*, instead of $\overline{a}_{\mathbf{r}}\langle y\rangle \mid \pi$, we may write $a(x,y) \mid x(*) \mid \pi$ which will delete whatever is linked to x. In order to simulate the superposition in (&), we may use contractions to obtain $?\Gamma_1$ and $?\Gamma_2$ from some $?\Gamma$; this is not the same as sharing some (possibly linear) Γ in the two sides of (&), but it works. Thus, there is a trade-off: more parallelism at the cost of duplication of resources when choice is encoded, but perhaps with careful stratification in the ordering of reductions (delay of contractions, priority to deletions), the effects can be minimised.

Complications arise with affinity and second-order features. For example, with $a[\mathsf{X}, b] \mid a(*) \rightsquigarrow \sim a \mid b(*)$, there is no $\mathsf{X}{=}B$ in the contractum, and typing would fail without (XW). For the same reason, we added $(\sim\mathsf{AX}{=})$.

3 Fixpoint Constructions and Recursive Linear Types

We introduce *replicated* stored terms that are as before, but contain a self-reference (exactly one for well-typed terms). Taking $\mathsf{fn}(\pi) = ac\vec{e}$:

$$?a(b) \mid !\overline{a}(\boldsymbol{\nu}c).\pi \rightsquigarrow ?\overrightarrow{e(x \otimes y)} \cdot (\pi\{^{b\vec{x}}/_{c\vec{e}}\} \mid !\overline{a}(\boldsymbol{\nu}c).\pi\{^{\vec{y}}/_{\vec{e}}\}) \qquad (?\mathsf{D}-\mu!)$$

$$?a(*) \mid !\overline{a}(\boldsymbol{\nu}c).\pi \overset{a}{\rightsquigarrow} \overrightarrow{?e(*)} \qquad (?\mathsf{W}-\mu!)$$

$$(\boldsymbol{\nu}\vec{e}\colon \vec{A})\,(?a(b \otimes d) \mid !\overline{a}(\boldsymbol{\nu}c).\pi \mid \rho) \qquad (?\mathsf{C}-\mu!)$$

$$\overset{a}{\rightsquigarrow} (\boldsymbol{\nu}\vec{e}\colon \vec{A})(\boldsymbol{\nu}\vec{x}\colon \vec{A})(\boldsymbol{\nu}\vec{y}\colon \vec{A})\left(?\overrightarrow{e(x \otimes y)} \;\middle|\; !\overline{b}(\boldsymbol{\nu}c).\pi\{^{b\vec{x}}/_{a\vec{e}}\} \;\middle|\; !\overline{d}(\boldsymbol{\nu}c).\pi\{^{d\vec{y}}/_{a\vec{e}}\} \;\middle|\; \rho\right)$$

Dereliction, $(?\mathsf{D}-\mu!)$, replicates the stored term, inducing a linear *unfolding*, which resembles the π-calculus equation $!P \equiv P \,|\, !P$. It follows that $(\boldsymbol{\nu}a)$ is only deallocated by weakening and contraction ($\rightsquigarrow$ vs. $\overset{a}{\rightsquigarrow}$).

Without any modification of types, we can replace promotion (P) with (recP):

$$\textit{Typing (for disambiguation):}\quad \vdash \pi \triangleright_{\boldsymbol{\mu}} \Gamma \qquad \frac{\vdash \pi \triangleright_{\boldsymbol{\mu}} ?\Gamma, a\colon ?A^{\perp}, b\colon A}{\vdash !\overline{a}(\boldsymbol{\nu} b\colon A).\pi \triangleright_{\boldsymbol{\mu}} ?\Gamma, a\colon !A}\ (\mathsf{recP})$$

In (recP), the (free) name a appears in π with a type that allows to perform a cut with (the copy of) $!\overline{a}\ldots$; now names may appear *three* times, but only one pair forms a cut at any moment. This kind of rule appears in the *Polarised* proof nets of Montelatici [33] but is not associated to exponentials, and in the linear λ-calculus `Lily` of Bierman et al. [10] where a similar approach to ours is taken (self-referencing !-terms), but in an intuitionistic setting.

The standard exponentials can be simulated, since the (unfolded) copy can be deleted after the box is opened for the first time: $!\overline{a}(\boldsymbol{\nu} x).(?a(*) \mid \pi)$. However, this comes at the cost of some useless copying, so it's not efficient. This extension integrates very well with the standard system, as a new *execution device* [21].

Recursive Types The previous solution is not perfect: once opened, replicated exponentials can only be *internally* reused but not returned. For example, $!\overline{a}(\boldsymbol{\nu} x\colon \mathbf{1} \otimes !A).\overline{x}(y,z)\cdot(\overline{y} \mid za)$ is not typable, as can be easily verified. In order to type such terms, reminiscent of objects with methods that return *self* [1], we introduce an original form of recursive linear types. Conveniently, the syntax of terms and the reduction relation need not change. Interfaces and types remain the same except that $!A$ and $?A$ are replaced by:

$$A, B, C ::= \cdot \cdot \mid \mathcal{X} \mid !\mathcal{X}.A \mid ?\mathcal{X}.A\,, \quad (!\mathcal{X}.A)^{\perp} \stackrel{\mathsf{def}}{=} ?\mathcal{X}.A^{\perp} \quad (?\mathcal{X}.A)^{\perp} \stackrel{\mathsf{def}}{=} !\mathcal{X}.A^{\perp} \quad \mathcal{X}^{\perp} \stackrel{\mathsf{def}}{=} \mathcal{X}$$

First, $\mathcal{X}$ is bound in $!\mathcal{X}.A$ and $?\mathcal{X}.A$. Moreover, there is no effective negation for $\mathcal{X}$ (in contrast to X), as it induces nonsensical dualities such as $(!\mathcal{X}.A \otimes \mathcal{X})^{\perp} \stackrel{\mathsf{def}}{=} ?\mathcal{X}.A^{\perp} \parr \mathcal{X}^{\perp}$. Type $!\mathcal{X}.A \otimes \mathcal{X}$ would unfold to $A \otimes (!\mathcal{X}.A \otimes \mathcal{X})$ and $?\mathcal{X}.A^{\perp} \parr \mathcal{X}^{\perp}$ to $A^{\perp} \parr (!\mathcal{X}.A \otimes \mathcal{X})$, due to the negative occurrence $\mathcal{X}^{\perp}$, so the types would not be dual any more. Also, we shall assume that recursive types are always *guarded*, which means that every occurrence of a type variable is separated by its binder by at least one non-recursive constructor, *e.g.*, $!\mathcal{X}.?\mathcal{Y}.\mathcal{X}$ is not guarded, but $!\mathcal{X}.A \otimes \mathcal{X}$ is. The isomorphism $!\mathcal{X}.(A \,\&\, B) \equiv !\mathcal{X}.A \otimes !\mathcal{X}.B$ does not hold; in fact the definability of one side does not imply that of the other, *e.g.*, $!\mathcal{X}.(\mathbf{1} \,\&\, \mathcal{X})$ is well-formed but $!\mathcal{X}.\mathbf{1} \otimes !\mathcal{X}.\mathcal{X}$ is clearly not, as $!\mathcal{X}.\mathcal{X}$ is not guarded.

Our formulation follows the *iso-recursive* approach, with folding performed in promotion and dereliction, and (simultaneous) unfolding effected with every cut as it connects the premises. The rules follow, recalling that there are no negative occurrences of $\mathcal{X}$, since $\mathcal{X}^{\perp}$ is by definition equal to $\mathcal{X}$, so unfolding substitutions have the intended effect. We omit the obvious ($\boldsymbol{\mu}\mathsf{W}, \boldsymbol{\mu}\mathsf{C}$).

$$\frac{\vdash \pi \triangleright_{\boldsymbol{\mu}} ?\Gamma, a\colon ?\mathcal{X}.A^{\perp}, b\colon A[!\mathcal{X}.A/\mathcal{X}]}{\vdash !\overline{a}(\boldsymbol{\nu} b\colon A[!\mathcal{X}.A/\mathcal{X}]).\pi \triangleright_{\boldsymbol{\mu}} ?\Gamma, a\colon !\mathcal{X}.A} \qquad \frac{\vdash \pi \triangleright_{\boldsymbol{\mu}} \Gamma, b\colon A[?\mathcal{X}.A/\mathcal{X}]}{\vdash ?a(b) \mid \pi \triangleright_{\boldsymbol{\mu}} \Gamma, b\colon [A[?\mathcal{X}.A/\mathcal{X}]], a\colon ?\mathcal{X}.A}\ (\boldsymbol{\nu}\mathsf{P},\ \boldsymbol{\mu}\mathsf{D})$$

When diverging terms can be defined, the system may be logically inconsistent, but we could introduce the recursive types in the language *without* replicated terms, and we conjecture that this preserves strong normalisation: all terms of recursive type will necessarily be finite approximations; see Baelde's work [7].

4 Soundness, Propositions-as-Types, and Progress

Due to space limitations, proofs are delegated to online appendices. Soundness (Theorem 1) holds for both the Linear/Affine Logic interpretations and for the recursive extension. With recursion, Static Correspondence (Theorem 2) becomes irrelevant, and Cut-elimination (Theorem 3) does not hold, but we obtain the weaker property of Cut-Progress (Theorem 4). To avoid confusion, we write $\vdash \pi \triangleright_\star \Gamma$ to indicate that a theorem holds for both $\vdash \pi \triangleright \Gamma$ and $\vdash \pi \triangleright_{\boldsymbol{\mu}} \Gamma$.

Theorem 1 (Soundness of $\equiv, \rightsquigarrow$). *(a) If $\vdash \pi \triangleright_\star \Gamma$ and $\pi \equiv \rho$ then $\vdash \rho \triangleright_\star \Gamma$ (b) If $\vdash \pi \triangleright_\star \Gamma$ and $\pi \rightsquigarrow \rho$ then $\vdash \rho \triangleright_\star \Gamma$*

Proof. By induction on the typing, and case analysis on the last $\rightsquigarrow$ rule applied. The absence of substitution is a simplifying factor, however, the presence of non-splitting links [21] — and note that in proof nets, cuts are actual links that we could denote with $\mathsf{cut}(b, c)$ — means that the last rule in a derivation cannot always be (Cut). For example, in the following the only reduction is a cut $xa \mid ay \overset{a}{\rightsquigarrow} xy$ but the last rule cannot be (Cut), so the cut on a is non-splitting.

$$\dfrac{\dfrac{\dfrac{}{\vdash xa \triangleright x\colon A^\perp, a\colon A}\,(\mathsf{Ax}) \qquad \dfrac{}{\vdash ay \triangleright a\colon A^\perp, y\colon A}\,(\mathsf{Ax})}{\vdash xa \mid ay \triangleright x\colon A^\perp, y\colon A, a\colon [A]}\,(\mathsf{Cut})}{\vdash z(x,y) \mid xa \mid ay \triangleright z\colon A^\perp \parr A, x\colon [A^\perp], y\colon [A], a\colon [A]}\,(\parr)$$

Similar situations arise with non-splitting instances of ($\otimes$). Our proof utilises several auxiliary definitions and lemmas that allow to conclude that there is an eventual subderivation with (Cut) even if it is not the last rule applied, without having to consider cases for each type rule within each case of $\rightsquigarrow$ (same for $\equiv$).

Let $\vdash \boldsymbol{\Gamma}$ denote a proof in Linear Sequent Calculus. The function $(\!|\cdot|\!)$ takes an interface Γ and returns the corresponding multiset of formulas $\boldsymbol{\Gamma}$, omitting discharged occurrences, *e.g.*, $(\!|a\colon A \otimes B, c\colon [B], b\colon C|\!) = A \otimes B, C$. Then:

Theorem 2 (Static Correspondence). *(a) If $\vdash \pi \triangleright \Gamma$ then $\vdash (\!|\Gamma|\!)$; and (b) if $\vdash \boldsymbol{\Gamma}$ then there exists π and Γ such that $\vdash \pi \triangleright \Gamma$ and $(\!|\Gamma|\!) = \boldsymbol{\Gamma}$.*

Proof. For part (a), we proceed by induction. For (b), we construct the δ-term (hence a proof net) that corresponds to a sequent proof Π of $\boldsymbol{\Gamma}$, by induction, as in Theorem 2.7 of the original article on Linear Logic [21].

Definition 1 (Cut-closed Typing). *An interface Γ is cut-closed when it does not contain discharged types or type variables, i.e., $\Gamma = \{a_i\colon A_i \mid i \in I\}$ where each A_i is a closed type. Let $\vdash \pi \triangleright^{\mathsf{c}} \Gamma$ mean that $\vdash \pi \triangleright \Gamma$ can be derived s.t. Γ is cut-closed, and similarly, let $\vdash \pi \triangleright^{\mathsf{c}}_{\boldsymbol{\mu}} \Gamma$ when $\vdash \pi \triangleright_{\boldsymbol{\mu}} \Gamma$ s.t. Γ is cut-closed.*

Theorem 3 (Cut-Elimination). *If $\vdash \pi \triangleright^{\mathsf{c}} \Gamma$, then there exists ρ s.t. $\vdash \rho \triangleright^{\mathsf{c}} \Gamma$, $\pi \rightsquigarrow^* \rho$, and there are no links in ρ (pairwise) sharing the same name as their conclusion (hence, there is no reducible cut in ρ). This is equivalent to stating that $\vdash \rho \triangleright^{\mathsf{c}} \Gamma$ is obtained without using the* (Cut) *rule.*

Proof. A complete proof of strong normalisation (sN) is beyond the scope of this work, but we can be confident since reduction is modelled precisely as cut-elimination in proof nets, and the equivalence $\equiv$ subsumes commutative contractions, i.e., it allows links that may react to be brought at the same level. Crucially, interfaces must be cut-closed: for every nested cut there is an outer scope (providing $\mathsf{G}[\,\cdot\,]$) to drive $\equiv$ using ($\equiv \mathsf{Split}\ \_$) rules. Moreover, in this case ($\equiv \mathsf{Res}\ \_$) are only needed for $\parr$-links; other cases are subsumed by ($\equiv \mathsf{Split}\ \_$).

For the system with recursion, we obtain a guarantee that cuts can always be effected even if it may be impossible to speak of cut-elimination.

Theorem 4 (Cut-Progress). *If $\vdash \pi \triangleright^{\mathsf{c}}_{\boldsymbol{\mu}} \Gamma$ and $\pi \not\rightsquigarrow$, then there are no links sharing the same name as their conclusion in π, except of the form $!\overline{a}(\boldsymbol{\nu} c\colon A).\rho$ with some $\mathsf{L}(a)$ in ρ. Equivalently, in the typing derivations of irreducible terms, only these lazy self-references may induce uses of* (Cut) *(i.e., within $\vdash \rho \triangleright_{\boldsymbol{\mu}} \Delta$).*

An open question We suspect that, with $\mathsf{G}[\,\cdot\,] \stackrel{\mathsf{def}}{=} \cdots \mid (\,\cdot\,)$, reduction modulo $\equiv$ is confluent, since normal forms that differ in the nesting structure of boxes, scopes, and parentheses can be equated. For example, with ($\equiv \mathsf{Split}\,\boldsymbol{\nu}$), $((\boldsymbol{\nu} bc)(a(b,c) \mid bc \mid \overline{e})) \equiv ((\boldsymbol{\nu} bc)(a(b,c) \mid bc) \mid \overline{e})$, knowing that $(\,\cdot\,)$ contains an independent sub-proof. For boxes, the result is suggested by the inclusion of ($\equiv ?\mathsf{C}!, \equiv ?\mathsf{CM}, \equiv ?\mathsf{AS}$) from Di Cosmo and Guerrini [13], and for the system with additives with our proposal of rule ($\equiv \&\&$); see Pagani and de Falco [35].

5 Parallel Translation of Polymorphic Functions

As an application, we define λ^2_δ, a linear variation of System F [24], and provide a sound, type-preserving translation to δ-calculus. This is of high interest since it allows to combine functional and process notations in the same programs (or proofs), and ultimately, to inform the compilation of functions into the highly parallel δ-calculus. The syntax is as follows, where p denotes *argument patterns*:

$$t, u \quad ::= \quad x \mid \lambda p.t \mid t\,u \mid \langle t, u\rangle \mid\ !t \mid \Lambda \mathsf{X}.t \mid t\,B$$

$$p \quad ::= \quad x^A \mid \langle x\rangle^A_{\mathbf{l}} \mid \langle x\rangle^A_{\mathbf{r}} \mid\ !x^A \mid x \circledcirc y^A \mid\ * \mid ()$$

$$\mathsf{let}\ p = u\ \mathsf{in}\ t \stackrel{\mathsf{def}}{=} (\lambda p.t)\,u$$

The notation is an obvious adaptation of δ-calculus, with a unit value $()$. We mostly omit type annotations, and do not consider multiplicative pairs, since $A \multimap B \multimap C \stackrel{\mathsf{iso}}{=} (A \otimes B) \multimap C$. Reduction applies in any context $\mathcal{C}$ [18].

$$(\lambda().t)\,() \to_\beta t \qquad (\lambda x.t)\,u \to_\beta t[u/x] \qquad (\Lambda \mathsf{X}.t)\,B \to_\beta t[B/X]$$

$$(\lambda\langle x\rangle_{\mathbf{l}}.t)\,\langle u_1, u_2\rangle \to_\beta t[u_1/x] \qquad (\lambda\langle x\rangle_{\mathbf{r}}.t)\,\langle u_1, u_2\rangle \to_\beta t[u_2/x] \qquad (\lambda !x.t)\,!u \to_\beta t[u/x]$$

$$(\lambda *.t)\,!u \to_\beta (\lambda *.(\cdots(\lambda *.t)\,x_1\cdots))\,x_n \quad \textit{with} \quad \mathsf{fv}(u) = x_1, \ldots, x_n$$

$$(\lambda x \circledcirc y.t)\,!u \to_\beta (\lambda m_n \circledcirc k_n.(\cdots(\lambda m_1 \circledcirc k_1.t[!u_1/x][!u_2/y])\,z_1\cdots))\,z_n$$

$$\textit{with} \quad \mathsf{fv}(u) = \vec{z}\,(= z_1, \ldots, z_n) \qquad u_1 = u[\vec{m}/\vec{z}] \qquad u_2 = u[\vec{k}/\vec{z}]$$

The exponential rules correspond closely to the reductions at the proof net/δ-calculus level. For example, $(\lambda *.t)\,!u$ induces weakenings for the free variables

of $!u$; similar exponential rules appear in $\lambda\mathsf{lxr}$, by Kesner and Lengrand [28]. In contraction, the type annotations of $\lambda m_i \mathbin{@} k_i$ can be recovered from the bindings of z_i on the surrounding context $\mathcal{C}$ (omitted here for brevity), as in δ-calculus.

Our original classical typing system for λ^2_δ is established on one-sided sequents of the form $\vdash [t]_a \triangleright \Gamma$ such that $a \notin \mathsf{fv}(t)$, where $[\,\cdot\,]_a$ is the *focus*, playing the role of the distinguished conclusion in intuitionistic typing. Interfaces and types remain as in δ-calculus, except for discharged conclusions which are not needed. For brevity, we do not show type annotations, and we freely use: $A \multimap B \stackrel{\mathsf{def}}{=} A^\perp \parr B$.

$$\frac{}{\vdash [x]_a \triangleright x : A, a : A^\perp} \qquad \frac{\vdash [t]_b \triangleright \Gamma, b : A \multimap B \quad \vdash [u]_c \triangleright \Delta, c : A}{\vdash [t\,u]_a \triangleright \Gamma, \Delta, a : B} \qquad (\mathsf{Ax},\ \mathsf{Cut})$$

$$\frac{\vdash [t]_b \triangleright \Gamma, x : A^\perp, b : B}{\vdash [\lambda x.t]_a \triangleright \Gamma, a : A \multimap B} \qquad \frac{\vdash [t]_b \triangleright \Gamma, b : A \qquad \vdash [u]_c \triangleright \Gamma, c : B}{\vdash [\langle t, u\rangle]_a \triangleright \Gamma, a : A \mathbin{\&} B} \qquad (\parr,\ \&)$$

$$\frac{\vdash [t]_b \triangleright \Gamma, y : B^\perp, b : C}{\vdash [\lambda\langle y\rangle_{\mathbf{l}}.t]_a \triangleright \Gamma, a : (A \mathbin{\&} B) \multimap C} \qquad \frac{\vdash [t]_b \triangleright \Gamma, x : A^\perp, b : C}{\vdash [\lambda\langle x\rangle_{\mathbf{r}}.t]_a \triangleright \Gamma, a : (A \mathbin{\&} B) \multimap C} \qquad (\oplus_{1,2})$$

$$\frac{\vdash [t]_b \triangleright \Gamma, x : A^\perp, b : B}{\vdash [\lambda !x.t]_a \triangleright \Gamma, a : (!A) \multimap B} \qquad \frac{\vdash [t]_b \triangleright \Gamma, b : B}{\vdash [\lambda *.t]_a \triangleright \Gamma, a : (!A) \multimap B} \qquad (\mathsf{D},\ \mathsf{W})$$

$$\frac{\vdash [t]_b \triangleright \Gamma, x : ?A^\perp, y : ?A^\perp, b : B}{\vdash [\lambda x \mathbin{@} y.t]_a \triangleright \Gamma, a : (!A) \multimap B} \qquad \frac{\vdash [t]_b \triangleright ?\Gamma, b : A}{\vdash [!t]_a \triangleright ?\Gamma, a : !A} \qquad (\mathsf{C},\ \mathsf{P})$$

$$\frac{\vdash [t]_b \triangleright ?\Gamma, b : A}{\vdash [\lambda().t]_a \triangleright ?\Gamma, a : \mathbf{1} \multimap A} \qquad \frac{}{\vdash [()]_a \triangleright a : \mathbf{1}} \qquad (\bot,\ \mathbf{1})$$

$$\frac{\vdash [t]_b \triangleright \Gamma, b\colon A \quad \mathsf{X} \notin \mathsf{ftv}(\Gamma)}{\vdash [\Lambda\mathsf{X}.t]_a \triangleright \Gamma, a\colon \forall\mathsf{X}.A} \qquad \frac{\vdash [t]_b \triangleright \Gamma, b\colon \forall\mathsf{X}.A}{\vdash [t\,B]_a \triangleright \Gamma, a\colon A[B/\mathsf{X}]} \qquad (\forall\mathsf{I},\ \forall\mathsf{E})$$

An example is $\vdash [\lambda !x.x]_a \triangleright a\colon !A \multimap A$. Observe that $!x$ always induces $\vdash [!x]_a \triangleright x : ?A^\perp, a : !!A$, so $\vdash [\lambda x.!x]_a \triangleright a\colon !A \multimap !!A$; this is called *digging*.

The located translation $[\![\cdot]\!]_a$ from λ^2_δ to δ is defined ignoring types, which can be recovered easily if we record the type of the location; also, in $[\![t]\!]_a$, $a \notin \mathsf{fv}(t)$.

$$[\![x]\!]_a = xa \qquad [\![()]\!]_a = \overline{a} \qquad [\![!t]\!]_a = !\overline{a}(\boldsymbol{\nu} c)\mathbin{.}[\![t]\!]_c \qquad [\![t\,u]\!]_a = (\boldsymbol{\nu} bcd)\,([\![t]\!]_b \mid \overline{b}(c,d) \mid da \mid [\![u]\!]_c)$$

$$[\![\lambda().t]\!]_a = (\boldsymbol{\nu} xb)\,(a(x,b) \mid x \mid [\![t]\!]_b) \qquad [\![\lambda x.t]\!]_a = (\boldsymbol{\nu} xb)\,(a(x,b) \mid [\![t]\!]_b)$$

$$[\![\lambda\langle x\rangle_{\mathbf{l}}.t]\!]_a = (\boldsymbol{\nu} xbz)\,(a(z,b) \mid \overline{z}_{\mathbf{l}}\langle x\rangle \mid [\![t]\!]_b) \qquad [\![\langle t,u\rangle]\!]_a = a_{\mathbf{l}}(\boldsymbol{\nu} b)\mathbin{.}[\![t]\!]_b + a_{\mathbf{r}}(\boldsymbol{\nu} c)\mathbin{.}[\![u]\!]_c$$

$$[\![\lambda\langle x\rangle_{\mathbf{r}}.t]\!]_a = (\boldsymbol{\nu} xbz)\,(a(z,b) \mid \overline{z}_{\mathbf{r}}\langle x\rangle \mid [\![t]\!]_b) \qquad [\![\lambda *.t]\!]_a = (\boldsymbol{\nu} zb)\,(a(z,b) \mid ?z(*) \mid [\![t]\!]_b)$$

$$[\![\lambda x \mathbin{@} y.t]\!]_a = (\boldsymbol{\nu} xyzb)\,(a(z,b) \mid ?z(x \mathbin{@} y) \mid [\![t]\!]_b) \qquad [\![\lambda !x.t]\!]_a = (\boldsymbol{\nu} xyb)\,(a(y,b) \mid ?y(x) \mid [\![t]\!]_b)$$

$$[\![\Lambda\mathsf{X}.t]\!]_a = (\boldsymbol{\nu} X)(\boldsymbol{\nu} b)\,(a[\mathsf{X},b] \mid [\![t]\!]_b) \qquad [\![t\,B]\!]_a = (\boldsymbol{\nu} bc)\,([\![t]\!]_b \mid \overline{b}[B,c] \mid ca)$$

The translation is remarkably parallel, *e.g.*, in $[\![\lambda p.t]\!]_a$, $[\![t]\!]_b$ is not prefixed, with the exception of $[\![\langle u,t\rangle]\!]_a$ and $[\![!t]\!]_a$, but this is exactly when laziness is useful. For example, $[\![\lambda x.!x]\!]_a = (\boldsymbol{\nu} xb)\,(a(x,b) \mid [\![!x]\!]_b)$ and $[\![(\lambda x.x)\,y]\!]_a \rightsquigarrow^* [\![y]\!]_a$. Translations of normal forms may be reducible: $[\![x\,y]\!]_a \rightsquigarrow^2 (\boldsymbol{\nu} cd)\,(yc \mid \overline{x}(c,d) \mid da)$. It is easy to verify that both $\vdash [x\,y]_a \triangleright \Gamma$ and $\vdash (\boldsymbol{\nu} cd)\,(yc \mid \overline{x}(c,d) \mid da) \triangleright \Gamma$, where $\Gamma = x\colon A \otimes B^\perp, y\colon A^\perp, a\colon B$. Similarly, $(\lambda x \mathbin{@} y.t)\,z \not\to_\beta$ but $[\![(\lambda x \mathbin{@} y.t)\,z]\!]_a \rightsquigarrow^* (\boldsymbol{\nu} xy)(?z(x \mathbin{@} y) \mid [\![t]\!]_a)$, and $[\![t]\!]_a$ may reduce further. (We used the proposition $[\![t]\!]_c \mid ca \overset{c}{\rightsquigarrow} [\![t]\!]_a$, which is trivial by induction on t.) As a last example: $(\lambda x \mathbin{@} y.t)\,!z \to^*_\beta (\lambda z_1 \mathbin{@} z_2.t[!z_1, !z_2/x, y])\,z$ and we obtain $[\![(\lambda x \mathbin{@} y.t)\,!z]\!]_a \rightsquigarrow^* (\boldsymbol{\nu} xyz_1z_2)(?z(z_1 \mathbin{@} z_2) \mid [\![!z_1]\!]_x \mid [\![!z_2]\!]_y \mid [\![t]\!]_a)$.

Recursion The syntax is extended with the new term $\mathsf{fix}\, x^A\, t$. The main novelty of reduction is an unfolding dereliction, where u_1 is obtained from $\mathsf{fix}\, y\, u$.

$$\boxed{\mathsf{fv}(u) \setminus y = \vec{z}\,(= z_1, \ldots, z_n)} \qquad (\lambda *.t)\, \mathsf{fix}\, y\, u \to_\beta (\lambda *.(\cdots(\lambda *.t)\, z_1 \cdots))\, z_n$$

$$(\lambda !x.t)\, \mathsf{fix}\, y\, u \to_\beta (\lambda m_n \mathbin{⊛} k_n.(\cdots(\lambda m_1 \mathbin{⊛} k_1.t[{}^{u_1}\!/_{x}])\, z_1 \cdots))\, z_n$$
$$\textit{with} \quad u_2 = u[{}^{\vec{k}}\!/_{\vec{z}}] \qquad u_1 = u[{}^{\vec{m}}\!/_{\vec{z}}][\mathsf{fix}\, y\, u_2/y]$$
$$(\lambda x \mathbin{⊛} y.t)\, \mathsf{fix}\, y\, u \to_\beta (\lambda m_n \mathbin{⊛} k_n.(\cdots(\lambda m_1 \mathbin{⊛} k_1.t[\mathsf{fix}\, y\, u_1/_{x}][\mathsf{fix}\, y\, u_2/y])\, z_1 \cdots))\, z_n$$
$$\textit{with} \quad u_1 = u[{}^{\vec{m}}\!/_{\vec{z}}] \qquad u_2 = u[{}^{\vec{k}}\!/_{\vec{z}}]$$

The system can be adapted as before, both with and without recursive types, omitting $(\boldsymbol{\mu}\lambda\mathsf{W}, \boldsymbol{\mu}\lambda\mathsf{C})$ which are trivial adaptations of (W, C):

$$\textit{Translation:} \quad [\![\mathsf{fix}\, x\, t]\!]_a = !\overline{a}(\boldsymbol{\nu} b).[\![t[{}^{a}\!/_{x}]]\!]_b \qquad \textit{Below:} \quad (\boldsymbol{\nu}\lambda\mathsf{P}, \boldsymbol{\mu}\lambda\mathsf{D})$$
$$\frac{\vdash [t]_b \triangleright_{\boldsymbol{\mu}} ?\Gamma, x\colon ?A^{\perp}, b : A}{\vdash [\mathsf{fix}\, x\, t]_a \triangleright_{\boldsymbol{\mu}} ?\Gamma, a : !A}\ (\mathsf{rec}\lambda\mathsf{P})$$
$$\frac{\vdash [t]_b \triangleright_{\boldsymbol{\mu}} ?\Gamma, x\colon ?\mathcal{X}.A^{\perp}, b\colon A[!\mathcal{X}.A/\mathcal{X}]}{\vdash [\mathsf{fix}\, x\, t]_a \triangleright_{\boldsymbol{\mu}} ?\Gamma, a\colon !\mathcal{X}.A} \qquad \frac{\vdash [t]_b \triangleright_{\boldsymbol{\mu}} \Gamma, x\colon A^{\perp}[{}^{?\mathcal{X}.A^{\perp}}\!/_{\mathcal{X}}], b\colon B}{\vdash [\lambda !x.t]_a \triangleright_{\boldsymbol{\mu}} \Gamma, a\colon (!\mathcal{X}.A) \multimap B}$$

Theorem 5 (Subject Reduction). *If* $\vdash [t]_a \triangleright \Gamma$ *and* $t \to_\beta u$ *then* $\vdash [u]_a \triangleright \Gamma$.

Theorem 6 (Correctness of $[\![\cdot]\!]_a$). *If* $\vdash [t]_a \triangleright \Gamma$, *then* $\vdash [\![t]\!]_a \triangleright \Gamma$.

Theorem 7 (Soundness of $[\![\cdot]\!]_a$). *If* $\vdash [t]_a \triangleright \Gamma$ *and* $t \to_\beta u$ *then* $[\![t]\!]_a \rightsquigarrow^* [\![u]\!]_a$.

Proof (Outline). From $[\![t]\!]_a[{}^{B}\!/_{\mathsf{X}}] = [\![t[{}^{B}\!/_{\mathsf{X}}]]\!]_a$, we obtain $[\![(\Lambda\mathsf{X}.t)\, B]\!]_a \rightsquigarrow^* [\![t[{}^{B}\!/_{\mathsf{X}}]]\!]_a$. Another typical case is: $[\![(\lambda x.t)\, u]\!]_a \rightsquigarrow^* (\boldsymbol{\nu} x)\, ([\![u]\!]_x \mid [\![t]\!]_a) \rightsquigarrow^* [\![t[{}^{u}\!/_{x}]]\!]_a$, where $[\![u]\!]_x$ resembles an explicit substitution. The second sequence follows by induction on t, cases on the occurrence of x, $(\equiv \mathsf{Split}\, \boldsymbol{\nu})$ to restrict the scope of x, then $[\![\cdot]\!]_a$.

Example 1 We can define lists of type $\mathsf{List}\, A$, adapting System F idioms [24]:

$$\mathsf{List}\, A \stackrel{\mathsf{def}}{=} \forall \mathsf{X}.!(A \multimap \mathsf{X} \multimap \mathsf{X}) \multimap \mathsf{X} \multimap \mathsf{X} \qquad \mathsf{nil} \stackrel{\mathsf{def}}{=} \Lambda\mathsf{X}.\lambda x^{!(A \multimap \mathsf{X} \multimap \mathsf{X})}.\lambda y^{\mathsf{X}}.\mathsf{let} * = x\ \mathsf{in}\ y$$
$$\mathsf{cons}\, u\, t \stackrel{\mathsf{def}}{=} \Lambda\mathsf{X}.\lambda x^{!(A \multimap \mathsf{X} \multimap \mathsf{X})}.\lambda y^{\mathsf{X}}.\mathsf{let}\, x_1 \mathbin{⊛} x_2 = x\ \mathsf{in}\ (\mathsf{let}\, !x_3 = x_1\ \mathsf{in}\ x_3\, u\, (t\, \mathsf{X}\, x_2\, y))$$
$$(u_1, u_2) \stackrel{\mathsf{def}}{=} \Lambda\mathsf{X}.\lambda x^{!(A \multimap \mathsf{X} \multimap \mathsf{X})}.\lambda y^{\mathsf{X}}.\mathsf{let}\, x_1 \mathbin{⊛} x_2 = x\ \mathsf{in}\ (\mathsf{let}\, !x_3 = x_1\ \mathsf{in}\ x_3\, u_1\, (x_2\, u_2\, y))$$
$$\textit{A 2-element list:} \quad (u_1, u_2)\, (\mathsf{List}\, A)\, \mathsf{cons}\, \mathsf{nil} \quad \to_\beta^* \quad \mathsf{cons}\, u_1\, (\mathsf{cons}\, u_2\, \mathsf{nil})$$

The cons function is copied (see $x_1 \mathbin{⊛} x_2$) and one copy (x_1) is derelicted (see $!x_3$) to build the inductive list with self-application. For $\vdash [u_i]_{b_i} \triangleright \Gamma_i, b_i\colon A$, we can obtain $\vdash [\mathsf{cons} u_1\, (\mathsf{cons} u_2\, \mathsf{nil})]_a \triangleright \Gamma_1, \Gamma_2, a\colon \mathsf{List}\, A$. The translations to δ-calculus are typed with the same interfaces, by Theorem 6, and are sound, by Theorem 7. It is an easy (albeit tedious) exercise to verify that $[\![\mathsf{cons} u_1\, (\mathsf{cons} u_2\, \mathsf{nil})]\!]_a$ contains no !/&-links, hence is completely parallel. Details are omitted due to space limitations.

Example 2 Let $\Omega \stackrel{\mathsf{def}}{=} \mathsf{fix}\, y\, (\lambda().\mathsf{let}\, !z = y\ \mathsf{in}\ z\, ())$. Then $\mathsf{let}\, !z = \Omega\ \mathsf{in}\ z\, () \to_\beta (\lambda().\mathsf{let}\, !z = \Omega\, \mathsf{in}\, z\, ())\, () \to_\beta \mathsf{let}\, !z = \Omega\, \mathsf{in}\, z\, () \to_\beta \cdots$, and $\vdash [\Omega]_a \triangleright_{\boldsymbol{\mu}} a\colon !\mathcal{X}.\mathbf{1} \multimap \mathcal{X}$. Similarly, $\vdash [\mathsf{let}\, !z = \Omega\ \mathsf{in}\ z\, ()]_b \triangleright_{\boldsymbol{\mu}} b\colon !\mathcal{X}.\mathbf{1} \multimap \mathcal{X}$. $[\![\Omega]\!]_a$ is left as an easy exercise.

Related work A System F encoding into Proof Nets has already been outlined in [21], using the graphical syntax. Abramsky [2] was the first to define a linear λ-calculus with explicit syntax for exponentials, but its big-step reduction does not directly match proof nets. The first-order languages closest to λ^2_δ are the linear λ-calculus of Ohta and Hasegawa [34], which uses explicit operators for exponentials, and $\lambda\mathsf{lxr}$ of Kesner and Lengrand [28], that uses explicit substitutions and is sound and complete for the intuitionistic fragment of proof nets. The correspondence of $\lambda\mathsf{lxr}$ is stronger than our own, since explicit substitutions allow a finer decomposition of β-reduction that enjoys completeness wrt proof net contractions [21]. Nevertheless, we believe that $\lambda\mathsf{lxr}$ has a direct translation to δ-calculus enjoying the same properties, and in particular we expect that the commuting conversions of $\lambda\mathsf{lxr}$ would arise naturally from $\equiv$. For linear λ-calculi with recursion see Bierman et al. [10] and Alves et al. [4]. Toninho et al. [40] translated λ-calculus into πDILL [11] (which we discuss in related work). Wadler translated CP [41] into a functional language with session types, where exponentials are restricted to function types (stable functions). In both [40,41] the translations are less parallel due to prefixes, *e.g.*, in [41], $\langle\!\langle \lambda x.t \rangle\!\rangle_a = a(x).\langle\!\langle t \rangle\!\rangle_a$.

6 Related Work and Conclusions

Abramsky [2] was the first to study the relationship between Linear Logic and processes, in the parallel language of *proof expressions* (PE). In this original interpretation the syntax is not π-calculus, and the constructs are close to proof net links, but the logical relationship is not as strong as in δ-calculus. For example, there are no commutative contractions, so reduction evaluates to *canonical* (i.e., lazy) and not *normal* forms, from which it follows that it does not enjoy cut-elimination, but the weaker property of *convergence*. Also, there is no scope restriction and therefore no bound names. Axioms are interpreted as variables, which explains why substitution is needed in PE, via rule (Cleanup), but not in δ-calculus. In PE there are explicit cut-links (*co-equations*) $t \bot u$, where t, u are terms. A tensor link $\overline{a}(b, c)$ is written as $a \bot (b \otimes c)$, and can interact with $a \bot (x \parr y)$ resulting in $b \otimes c \bot x \parr y$ and then $b \bot x, c \bot y$. The $\oplus$-link $a \bot \mathsf{inl}(t)$ defines a prefix on t, but in δ-calculus it maps to $\overline{a}_{\mathsf{l}}\langle b \rangle \mid [\![b \bot t]\!]$, which is more parallel.

Bellin and Scott [9] studied a π-calculus *translation* of linear proofs, defined in unpublished lectures by Abramsky; in fact, Milner had also obtained a similar translation, albeit never published [3]. In this interpretation each connective translates to a composition of binding, sequencing, and input (resp. output). For instance, a sequent proof with a $\otimes$-conclusion maps to a term $(\boldsymbol{\nu} xy)\, \overline{a}\langle x, y \rangle.(P \mid Q)$, one with a $\parr$-conclusion becomes $a(x, y).P$, and axioms are *forwarders*, $a(x).b\langle x \rangle$. Therefore, it represents a regression wrt parallelism due to π-calculus sequentiality and non-local substitution. Still, Abramsky's translation identified a canonical process-algebraic correlate for the symmetric cut, in the form of parallel composition under name restriction. The exponentials were given a complicated encoding using summation, which indicates that typing with Linear Logic would be problematic, as summation would also map to the &-rule.

Recently, Honda and Laurent formalised a close correspondence between *Polarised* Proof Nets and π-calculus [25]. However, the variety of proof nets that they used consist solely of exponentials and the type system is not Linear Logic. Nevertheless, they illuminate the relationship with the non-deterministic behaviours of π-calculus, which may inform extensions of δ-calculus. Ehrhard and Laurent [17] presented a very close correspondence between the Acyclic Solos calculus and the Differential Interaction Nets of Ehrhard and Regnier [16], which is *not* typed with Linear Logic, but shares with δ-calculus the notion of binary actions (solos) corresponding to the multiplicative links; however, it does not cover additive, exponential, and quantification connectives. In preliminary work by Laneve, Parrow, and Victor [30], Solo diagrams were briefly shown to be related to untyped, multiplicative proof nets. Indeed, at this level Solos is our closest relative, due to the absence of prefix, but it does not support summation, it lacks explicit axioms, and it employs a standard structure congruence that, as we discussed, would not preserve soundness in a linear typing.

Caires and Pfenning [11] (with Toninho in [12]) interpreted sessions [39,26] in a variation of Intuitionistic Linear Logic, and obtained a propositions-as-types correspondence, using an almost standard π-calculus (πDILL) as the representation of *sequent proofs*. In this system the output $\overline{x}\langle y\rangle.P$ induces $x\colon A \otimes B$. This interpretation is *sequential*: first send some A on x, then do some B on x. Operationally, whenever sessions are interleaved only one of them can immediately reduce. For example, the πDILL term: $(\boldsymbol{\nu} xy)\,(\,\overline{x}\langle a\rangle.(\overline{y}\langle b\rangle.P \mid \overline{x}\langle m\rangle.Q) \mid x(c).R \mid y(k).S\,)$ must first react on x, after which both x, y may have independent redices. Session variables are not strictly linear, e.g., here $\overline{x}\langle\cdot\rangle$ appears twice, which is another manifestation of a causal relationship that cannot be safely broken: in typing the output $\overline{x}\langle a\rangle\ldots, x$ is not only a conclusion, but also a premise (along with a) from the usage $\overline{x}\langle m\rangle.Q$ under the prefix, and likewise for any x in Q. Parallelism is limited since *every* action (a prefix) forces a sequentialisation (a *box*), and reduction corresponds to proof normalisation of intuitionistic sequent proofs. Also, in [12] axioms $a \leftrightarrow x$ are interpreted as forwarders (in [11] axioms are not interpreted). Specifically, $(\boldsymbol{\nu} x)\,(P \mid a \leftrightarrow x) \rightarrow P[a/x]$, which requires substitution.

Recently Wadler [41] presented CP, an adaptation of πDILL that interprets sequent proofs of Classical Linear Logic. Reduction is defined as cut-elimination at the sequent proof level, prefixes (hence boxes) are preserved, and therefore there is not benefit wrt parallelism. The second-order quantification which is missing from [11,12] is added, under an interpretation in which a type is communicated, following the tradition of System F. Commutations are defined on *type derivations* and not on terms, which seems a highly inefficient method during reduction. As in πDILL, axioms are forwarders, and substitution is required.

Very recently, DeYoung et al. [15] developed a variation of πDILL based on asynchronous π-calculus, using a binary output $\overline{a}\langle b, c\rangle$ ($\otimes$-typed), and prefixed input $a(b, c).P$ ($\parr$-typed), together with constructs for additives that are similar to those of δ-calculus. This work was developed independently from ours, and we briefly explain some differences. First, it interprets *intuitionistic sequent proofs* and not proof nets, although in a sense it seems to go half-way

towards them, by allowing an asynchronous interpretation of $\otimes$, but it does not allow the same for $\parr$. As a result, parallelism is still limited: in δ-calculus the term $\overline{a}(b,c) \mid bx \mid cy \mid z(x,y)$ has a *non-splitting* $\otimes$-link, which means that we cannot obtain a type derivation (or sequent proof) with $(\otimes)$ as the last rule, and we are forced to start with the $\parr$-link. Still, a reduction on a can be performed in parallel to one on z. In [15], the input $z(x,y)$ necessarily appears as a prefix: $z(x,y).(\overline{a}\langle b,c\rangle \mid b \leftrightarrow x \mid c \leftrightarrow y)$, and cannot react at the same time as $\overline{a}\langle b,c\rangle$, although it would be perfectly safe. Moreover, no commutation can bring the output before the input, as this would be untypable (non-splitting), even if other terms can enter under the prefix. Beffara [8] also presented a synchronous π-calculus with $(\otimes/\parr)$ type rules for binary prefixes.

Beyond the issues regarding parallelism, none of the above works have provisions for affine typing or recursion. We plan to characterise *correctness* [23] and proof net equality using process-algebraic techniques, and to investigate concurrency extensions, *e.g.*, sharing, in order to encode references and full π-calculus.

Acknowledgements The author would like to thank Vasco T. Vasconcelos for his extensive comments, as well as the reviewers of previous versions.

References

1. Martín Abadi and Luca Cardelli. *A Theory of Objects*. Springer-Verlag, 1996.
2. Samson Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 111:3–57, 1993.
3. Samson Abramsky. Proofs as processes. In *Theoretical Computer Science*, 1994.
4. Sandra Alves, Maribel Fernández, Mário Florido, and Ian Mackie. Linearity and recursion in a typed lambda-calculus. PPDP '11, pages 173–182. ACM, 2011.
5. Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2:297–347, 1992.
6. Andrea Asperti and Luca Roversi. Intuitionistic light affine logic. *ACM Transactions on Compututational Logic*, 3(1):137–175, January 2002.
7. David Baelde. Least and greatest fixed points in linear logic. *ACM Trans. Comput. Logic*, 13(1):2:1–2:44, January 2012.
8. Emmanuel Beffara. A concurrent model for linear logic. *ENTCS*, 2006.
9. G. Bellin and P.J. Scott. On the pi-calculus and linear logic. *Theoretical Computer Science*, 135(1):11 – 65, 1994.
10. G. M. Bierman, A. M. Pitts, and C. V. Russo. Operational properties of Lily, a polymorphic linear lambda calculus with recursion. In *HOOTS*, September 2000.
11. Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In *Proceedings of CONCUR'10*, pages 222–236. Springer-Verlag, 2010.
12. Luís Caires, Frank Pfenning, and Bernardo Toninho. Towards concurrent type theory. In *TLDI '12*. ACM, 2012.
13. Roberto Di Cosmo and Stefano Guerrini. Strong normalization of proof nets modulo structural congruences. RtA '99, 1999.
14. Vincent Danos and Laurent Regnier. The Structure of Multiplicatives. *Archive for Mathematical Logic*, 28:181–203, 1989.
15. H. DeYoung, L. Caires, F. Pfenning, and B. Toninho. Cut Reduction in Linear Logic as Asynchronous Session-Typed Communication. In *CSL'12*, 2012.

16. T. Ehrhard and L. Regnier. Differential interaction nets. *Theoretical Computer Science*, 364:166–195, November 2006.
17. Thomas Ehrhard and Olivier Laurent. Acyclic solos and differential interaction nets. *Logical Methods in Computer Science*, 6(3), 2010.
18. Matthias Felleisen and Robert Hieb. The revised report on the syntactic theories of sequential control and state. *Theoretical Computer Science*, 103(2):235–271, 1992.
19. Philippa Gardner, Cosimo Laneve, and Lucian Wischik. Linear forwarders. *Information and Computation*, 205(10):1526–1550, 2007.
20. Philippa Gardner and Lucian Wischik. Explicit fusions. In *MFCS 2000*, volume 1893 of *LNCS*, pages 373–382. Springer-Verlag, 2000.
21. Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
22. Jean-Yves Girard. Linear logic: its syntax and semantics. In *Advances in Linear Logic*, pages 1–42. Cambridge University Press, 1995.
23. Jean-Yves Girard. Proof-nets: The parallel syntax for proof-theory. In *Logic and Algebra*, pages 97–124. Marcel Dekker, 1996.
24. Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and types*. Cambridge University Press, New York, NY, USA, 1989.
25. Kohei Honda and Olivier Laurent. An exact correspondence between a typed pi-calculus and polarised proof-nets. *Theoretical Comp. Science*, 411:223–238, 2010.
26. Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. Language Primitives and Type Disciplines for Structured Communication-based Programming. In *ESOP'98*.
27. D.J.D. Hughes and R.J. van Glabbeek. Proof nets for unit-free multiplicative-additive linear logic. *ACM Trans. on Computational Logic*, 6(4):784–842, 2005.
28. Delia Kesner and Stéphane Lengrand. Resource operators for λ-calculus. *Information and Computation*, 205(4), April 2007.
29. Yves Lafont. Interaction combinators. *Inf. and Computation*, 137:69–101, 1995.
30. Cosimo Laneve, Joachim Parrow, and Björn Victor. Solo diagrams. In *TACS '01*, volume 2215 of *LNCS*, pages 127–144. Springer-Verlag, 2001.
31. Cosimo Laneve and Björn Victor. Solos in concert. *Mathematical Structures in Computer Science*, 13(5):657–683, October 2003.
32. Robin Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2(2):119–141, 1992.
33. Raphaël Montelatici. Polarized proof nets with cycles and fixpoints semantics. TLCA'03, pages 256–270, Berlin, Heidelberg, 2003. Springer-Verlag.
34. Yo Ohta and Masahito Hasegawa. A terminating and confluent linear lambda calculus. RTA'06, 2006.
35. Michele Pagani and Lorenzo Tortora de Falco. Strong normalization property for second order linear logic. *Theoretical Computer Science*, 411(2), January 2010.
36. Joachim Parrow and Björn Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *LICS '98*. Computer Society Press, 1998.
37. Jorge A. Pérez, Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logical relations for session-based concurrency. In *ESOP '12*, pages 539–558, 2012.
38. Davide Sangiorgi and David Walker. *The pi-calculus, a theory of mobile processes*. Cambridge University Press, 2001.
39. Kaku Takeuchi, Kohei Honda, and Makoto Kubo. An Interaction-based Language and its Typing System. In *PARLE '94*, pages 398–413. Springer-Verlag, 1994.
40. Bernardo Toninho, Luís Caires, and Frank Pfenning. Functions as session-typed processes. In *FoSSaCS '12*, pages 346–360, 2012.
41. Philip Wadler. Propositions as sessions. In *Proceedings of the International Conference on Functional Programming (ICFP '12)*. ACM, 2012.